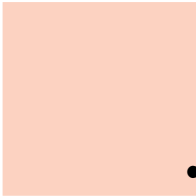


● ● ● ● ● ●

51

•
•
•
•
•
•

Apéndice2.5: Medida del tiempo de retardo	115
Simulación temporal del circuito	115
Frente de onda para la estimulación	117
Apéndice2.6: Package	121
Apéndice2.7: Multiplexor y decodificador	123
Multiplexor	123
Decodificador	124
Decodificador - multiplexor	125
Puerta de tres estados	126
Constructores condicionales de asignación de señal en VHDL	127
Sentencia de asignación de señal concurrente simple	127
Asignación de una señal de forma condicional	130
Selección en la asignación de una señal	136



Práctica 2

Sumador de 4 bits

.....

En esta sesión se trata de consolidar los conocimientos adquiridos en la sesión anterior, a la vez que se presentan nuevas funcionalidades de las herramientas Quartus y Modelsim y del lenguaje VHDL.

Respecto a Quartus, una de ellas es la utilización de librerías de usuario. Otras son la utilización de buses¹ y los pasos que hay que efectuar para asociar programas de prueba a diseños. Respecto a Modelsim, una de las nuevas funcionalidades es la observación de buses.

Respecto a VHDL, se describe cómo construir diseños jerárquicos y la especificación de buses. Estas funcionalidades son extremadamente útiles en la construcción de circuitos complejos, que usualmente se describen de forma estructural. Además, se muestra cómo parametrizar diseños y cómo instanciar e interconectar réplicas de un elemento mediante un constructor del lenguaje. Por otro lado, se introduce la utilización de programas de prueba, escritos en VHDL, para comprobar o verificar si el funcionamiento de un diseño se adecua a las especificaciones².

En este sentido se construirá y simulará un **sumador de 4 bits con propagación serie del acarreo**, utilizando como módulo básico el **sumador de 1 bit (full adder)** diseñado en la sesión anterior.

Por último, respecto al lenguaje VHDL, se introduce la descripción de un modelo de comportamiento mediante la sentencia “process”, que permite efectuar una especificación secuencial, de forma similar a un lenguaje de programación clásico, para describir el comportamiento de un sistema en función del tiempo. En esta práctica este método de descripción lo utilizaremos para diseñar programas de prueba³.

-
1. Un bus es un conjunto de señales agrupadas, que tienen relación entre sí y que son del mismo tipo.
 2. El número de ingenieros que trabaja en la verificación de un diseño es más del doble de los que trabajan en una descripción VHDL.
 3. En la siguiente sesión se utilizará para describir circuitos secuenciales.

En un apéndice se describe la especificación en VHDL, utilizando constructores condicionales de asignación de señal, de componentes básicos en el diseño de circuitos combinatoriales como son el multiplexor, el decodificador y una puerta de tres estados.

Diseño de un sumador de 4 bits con propagación serie del acarreo

Para representar un número natural (\underline{a}) se utiliza un vector de bits $A = (a_{n-1}, \dots, a_1, a_0)$ que se interpreta de forma ponderada.

$$\underline{a} = \sum_{i=0}^{n-1} a_i \times 2^i$$

donde \underline{a} es el valor numérico que se calcula como la suma ponderada de los bits del vector de bits.

Expresiones algebraicas. Dados los vectores de bits (A, B) de entrada

$$A = (a_{n-1}, \dots, a_1, a_0) \quad B = (b_{n-1}, \dots, b_1, b_0)$$

que representan los números naturales \underline{a} y \underline{b} respectivamente, la operación suma se expresa como $\underline{s} = (\underline{a} + \underline{b}) \bmod 2^n$. El resultado \underline{s} se representa mediante un vector de bits $S = (s_{n-1}, \dots, s_1, s_0)$.

La suma de los vectores A y B se efectúa sumando bit a bit los vectores de bits y propagando el acarreo.

$$a_i + b_i + c_i = 2 \times c_{i+1} + s_i \quad 0 \leq i < n \quad c_0 = 0$$

Condición de irrepresentabilidad. Si la suma $\underline{a} + \underline{b}$ es mayor que $2^n - 1$ el resultado no se puede representar con n bits. La función de irrepresentabilidad es: $\text{Irre} = c_n$.

Modelo de comportamiento en VHDL

En esta sección, después de describir nuevas funcionalidades de VHDL, tales como la especificación de grupos de señales o buses, se describe un modelo de comportamiento para un sumador de 4 bits.

Tipos de datos en VHDL

Cada objeto VHDL debe tener asociado un tipo. La noción de tipo en VHDL es clave puesto que es un lenguaje fuertemente tipado, que requiere que cada objeto sea de un cierto tipo. VHDL incluye varios tipos predefinidos y permite al usuario que defina sus propios tipos según los necesite.

Ejemplo	Ejemplo
signal bus: std_logic_vector (3 downto 0);	signal bus: std_logic_vector (0 to 3);
Elemento a elemento	Elemento a elemento
d(3) <= '0';	d(0) <= '1';
d(2) <= '1';	d(1) <= '0';
d(1) <= '0';	d(2) <= '1';
d(0) <= '1';	d(3) <= '0';
Todos los bits a la vez	Todos los bits a la vez
d <= "0101";	d <= "1010";

Elementos léxicos en VHDL

Recordemos que la definición de valores en el tipo `std_logic` se ha efectuado mediante el tipo carácter. Puesto que el tipo carácter se utiliza en la definición, la forma de especificar los valores es parte de la definición y deben de especificarse de la misma forma. Por ello deben de especificarse con mayúsculas.

Caracteres. Un carácter se especifica entre comillas: Por ejemplo, los valores de tipo `std_logic` se especifican como: '0', '1', 'U', 'X', '-'.

Cadenas de caracteres. Una cadena de caracteres se especifica entre dobles comillas: Por ejemplo, el valor de una señal de 5 bits de tipo `std_logic_vector` se especifica como: "01UX-".

Los vectores de bits constantes se especifican como una cadena de caracteres.

Una cadena de bits representa una secuencia de bits. Para indicar que es una cadena de bits se utiliza el prefijo B. Las cadenas de caracteres también pueden interpretarse en hexadecimal y en octal. En estos dos últimos casos el prefijo es X y O respectivamente. Por ejemplo, en binario tendríamos B"101010", en hexadecimal X"B6" y en octal O"135".

Tengamos en cuenta que en hexadecimal un dígito representa 4 bits. En consecuencia el número B"101" es distinto del número X"5" puesto que en el primero sólo se utilizan 3 bits mientras que el segundo representa una secuencia de 4 bits. Casos similares deben tenerse en cuenta en octal.

Un vector en el que todos los bits tienen el mismo valor puede especificarse mediante la palabra clave "others" de la siguiente forma⁴.

4. Esta forma es un caso particular de agregación "aggregate": utilizando la palabra clave "others" no es necesario conocer el tamaño del vector. Una agregación es un grupo de valores que se utiliza para establecer valores en un vector. Un "aggregate" debe establecer un valor en todos los elementos. Hay dos formas de hacerlo: a) posición o b) asociación de nombre.

Ejemplo

```
signal bus: std_logic_vector (3 downto 0) ;  
bus <= (others => '0'); -- es lo mismo que bus <= "0000"
```

Operadores

Además de los operadores lógicos comentados en la sesión anterior se dispone de otros operadores.

Operadores aritméticos. +, - y * entre otros. El resultado que se obtiene es del mismo tipo que los operandos.

Para utilizar los operadores previos es necesario hacer visible el “package” `numeric_std.all` de la librería IEEE. En este package se definen los tipos `unsigned` y `signed` basados en el tipo `std_logic` y se definen las operaciones aritméticas. El tipo `unsigned` se utiliza para identificar números naturales en binario y el tipo `signed` se utiliza para identificar número enteros en complemento a dos. En los dos casos, el bit más significativo es el de la izquierda (mayor ponderación). Tengamos en cuenta que el resultado de una operación depende del tipo. En el Apéndice 2.1 se detalla la conversión y asimilación de tipos cuando se utiliza el “package” `numeric_std.all` de la librería IEEE.

Tipos		Asimilación	
Fuente	Destino	de tipo	Ejemplo
std_logic_vector	signed	signed ()	signal uns: unsigned (4 downto 0);
std_logic_vector	unsigned	unsigned ()	signal stdlv: std_logic_vector (4 downto 0);
signed	std_logic_vector	std_logic_vector ()	. . .
unsigned	std_logic_vector	std_logic_vector ()	uns <= unsigned (stdlv);

Otro operador permite concatenar vectores de bits.

Operador de concatenación (“Concatenate”, &) de bits o vectores de bits. Junta en un vector de bits dos vectores de bits o un bit y un vector de bits.

Mediante este último operador se pueden especificar los valores de todos los bits de una señal a partir de señales que tienen menos bits o subconjuntos de bits de señales.

Ejemplo

```

signal bus: std_logic_vector (3 downto 0);
signal destino: std_logic_vector (4 downto 0);
destino <= bus(3) & bus (3 downto 1) & '1';

```

Modelo de comportamiento de un sumador de 4 bits

En la Figura 2.1 se muestra un esquema del circuito. Las señales A y B son grupos de señales de 4 bits. Esto es, los vectores de bits utilizados en las expresiones algebraicas descritas previamente. La señal de salida SUM también es un grupo de 4 señales.

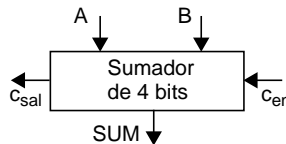


Figura 2.1 Módulo sumador de 4 bits.

Un sumador de 4 bits puede especificarse en VHDL utilizando un modelo de comportamiento y sentencias de asignación de señales concurrentes (Figura 2.2).

Notemos que en la especificación del sumador de 4 bits se utiliza una señal interna con un bit más (mayor rango de valores) para disponer del acarreo de salida (Figura 2.2). La operación de suma se efectúa con vectores de 5 bits. Para ello se incrementa el número de bits con el que se representan los datos de entrada. En el caso de A y B se añade un bit, mientras que en el caso de cen se añaden 4 bits a la izquierda. Como interpretamos los vectores de bits como números naturales (unsigned), el valor de los bits añadidos es cero. En estas condiciones, el acarreo de salida es el bit más significativo de la señal SINTER y el valor de la suma está representado por los bits restantes.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity S4bits is
port (
    A: in      std_logic_vector (3 downto 0) ;
    B: in      std_logic_vector (3 downto 0) ;
    cen: in     std_logic;
    SUM: out    std_logic_vector (3 downto 0) ;
    csal: out   std_logic);
end S4bits;

architecture comportamiento of S4bits is
-- definición de una suma interna (SINTER) que tiene en cuenta el acarreo de salida
constant retardo: time := 60 ns;
signal SINTER: std_logic_vector (4 downto 0);
begin
    SINTER <= std_logic_vector(('0' & unsigned(A)) + ('0' & unsigned(B)) + ("0000" & cen)) after retardo;
    csal <= SINTER(4);
    SUM <= SINTER (3 downto 0);
end comportamiento;
```

Figura 2.2 Modelo de comportamiento en VHDL de un sumador de 4 bits.

En este ejemplo se especifica un retardo que es independiente del valor de los vectores de bits que se suman.

En el esquema de la Figura 2.4, teniendo en cuenta el retardo de las puertas, la propagación del acarreo determina el retardo de la suma de los vectores de bit.

Se tardan 30 ns en calcular la señal s_0 y 25 ns en calcular la señal c_1 . En paralelo se ha calculado la señal $a_1 \oplus b_1$. Entonces, después de calcular c_1 , el cálculo de la señal s_1 experimenta el retardo de una puerta XOR y el cálculo de la señal c_2 experimenta el retardo de una puerta AND y una puerta OR. Por tanto, el retardo de este sumador de 4 bits es $25 \times 4 = 100$ ns.

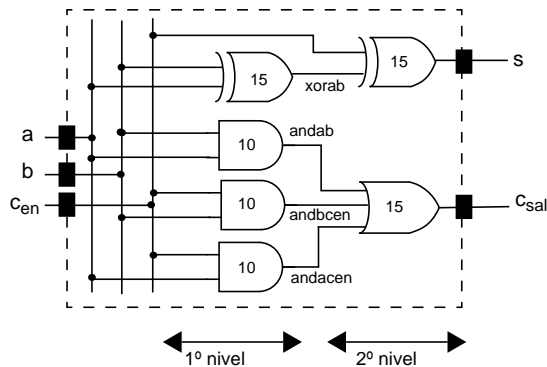


Figura 2.5 Sumador de 1 bit.

Diseño jerárquico o estructural en VHDL

A medida que los diseños son más complejos (incluyen más elementos) es necesario estructurar el diseño de forma modular. Ello ayuda a que el diseño se comprenda mejor, ya que se encapsulan detalles de implementación de algunas funcionalidades. Además, los módulos que implementan las funcionalidades pueden reutilizarse en otros diseños o refinar para el diseño que nos ocupa. Por otro lado, la verificación del correcto funcionamiento de un módulo concreto se puede efectuar de forma aislada e independiente de la interacción con otros módulos, lo cual facilita la tarea.

En el lenguaje VHDL se pueden describir modelos jerarquizados mediante lo que se denomina un modelado estructural.

Modelo estructural (“Structural model”). Se especifican los elementos o componentes que se utilizan y cómo se interconectan.

En la descripción de un modelo estructural, cada componente o elemento se ha definido previamente y puede haber sido descrito utilizando un modelo estructural o un modelo de flujo de datos (caso particular de modelo de comportamiento).

Una descripción estructural puede compararse a un diagrama esquemático como el de la Figura 2.4. Esto es, un esquema de bloques del circuito en el cual se describen los componentes y su interconexión. Por ello, es útil partir del esquema de circuito para efectuar la descripción VHDL.

Recopilando, una descripción estructural se expresa en términos de subsistemas interconectados por señales. Cada subsistema puede a la vez estar expresado por una interconexión de subsistemas y así de forma sucesiva, hasta que se llega a un nivel de componentes “primitivos” descritos puramente en términos de comportamiento.

En VHDL la forma de describir un diseño estructural es:

- Especificación de la interfaz.
- Declaración de la lista de componentes que se van a utilizar.
- Declaración de las señales que definen los cables que interconectan los componentes. Esto es, las señales internas.
- Instanciación y etiquetado de cada instancia de un mismo componente, de forma que cada instancia está unívocamente definida. Interconexión de las instancias.

Como ejemplo conductor utilizaremos el circuito de la Figura 2.4, que puede describirse utilizando sumadores de 1 bit como elementos.

El primer paso es independiente del tipo de modelado utilizado en VHDL para describir el funcionamiento de un módulo. La declaración de la interfaz se obtiene a partir de las señales de entrada y salida del módulo que se quiere modelar. La interface del sumador de la Figura 2.4 es la interface de la Figura 2.2 y si se utilizan parámetros es semejante a la interface de la Figura 2.3.

Declaración de componentes y señales internas

Antes de instanciar los componentes deben ser declarados en: a) la sección de declaraciones de la arquitectura, o b) en la declaración de “packages” que se utilizan en la cabecera del fichero. En VHDL esta declaración indica que los módulos están disponibles o accesibles para ser usados en el diseño que estamos especificando. Esta acción es la que describe el diseño de forma jerárquica.

Las señales de interconexión entre los módulos (señales internas), se declaran dentro del cuerpo de la arquitectura, después de la palabra clave “architecture” y antes de la palabra clave “begin” (Figura 2.6).

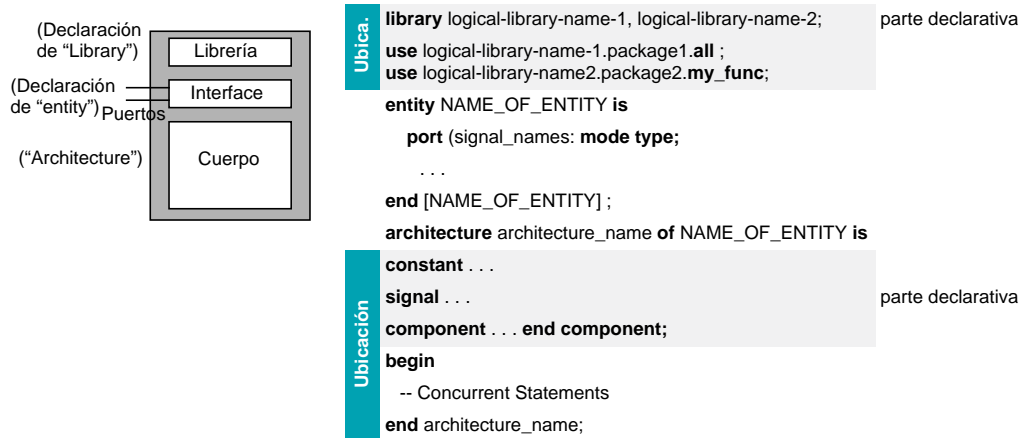


Figura 2.6 Partes declarativas.

Seguidamente se describe la opción a) de declaración de componentes en la arquitectura (instanciación indirecta de entidades mediante componentes). La opción de instanciación directa de entidades se describe en el Apéndice 2.1.

Declaración de los componentes en la arquitectura. Los componentes se declaran dentro del cuerpo de la arquitectura, siguiendo a la declaración de ésta. Esto es, después de la palabra clave "architecture" y antes de la palabra clave "begin" (Figura 2.6).

La forma de efectuar la declaración de los módulos que se van a utilizar es describir su interface. En este caso se utiliza la palabra clave "component". Seguidamente se especifican las señales de entrada y salida, de la misma forma que se ha efectuado en la declaración de la interface ("entity") al describir el módulo. La declaración finaliza con las palabras clave "end component". En resumen, la declaración de un componente consiste del nombre del componente y de la interface (puertos), siendo la sintaxis la siguiente.

Sintaxis

```

component component_name
[ port (port_signal_name: mode type;
    port_signal_name: mode type;
    ...
    port_signal_name: mode type); ]
end component;
  
```

Declaración de la ubicación (localización del componente). El nombre de un componente se refiere al nombre de una entidad definida explícitamente en un fichero VHDL. La lista de puertos de la interface especifica el nombre, el modo y tipo de cada puerto de forma idéntica a como se especifica en la declaración "entity". En el fichero donde se declaran los componentes hay que indicar que el fichero que contiene la especificación del componente se almacena en el directorio de trabajo. Esta especificación se efectúa en la cabecera del fichero⁶.

Sintaxis

```
use work.all;
```

Sintaxis

```
use work.my_component.all;
```

En la especificación mostrada a la izquierda, se indica que son accesibles todas las especificaciones existentes en el directorio de trabajo. En la especificación de la derecha, se indica que sólo son accesibles las especificaciones incluidas en el fichero denominado my_component.vhd.

Instanciación de componentes e interconexión

El último paso al crear un modelo estructural es crear las instancias de los módulos que se utilizan en el diseño y describir su interconexión. En la interconexión se efectúa una asociación (mapeo) entre las señales de la interfaz de los módulos, que se utilizan como componentes, y las señales de entrada, salida e internas del diseño que nos ocupa. Estamos asociando señales en un nivel de la jerarquía del diseño con señales definidas en un nivel previo de la jerarquía (puertos).

La instanciación de componentes se efectúa después de la palabra clave “begin” en el cuerpo de la arquitectura.

```
begin
-- Component instantiation and connections
-- Concurrent Statements
end architecture_name;
```

Una sentencia de instanciación de componente es una sentencia concurrente. El orden en el cual se instancian los componentes no es importante desde el punto de vista de la evaluación (simulación), puesto que las sentencias son concurrentes y por tanto se evalúan en paralelo. Esto es, el esquema de circuito que se describe mediante las sentencias es el mismo, independientemente del orden en que se especifiquen.

Cada instancia de un módulo tiene un nombre que la identifica de forma unívoca y está seguida por el nombre del componente. El nombre del componente precede a la palabra clave “port map”, a partir de la cual se especifica la asociación de señales entre niveles de la jerarquía. La sintaxis para la instanciación de un componente es la siguiente.

Sintaxis

```
instance_name: [component] component_name
port map (port1 => signal1, port2 => signal2, . . . , portn => signaln);
```

6. Existen otras opciones que no utilizaremos por ahora.

Sumador de 4 bits

En primer lugar se especifica el nombre de la instancia seguido de dos puntos y el nombre del componente y la palabra clave “port map”. El nombre de la instancia o etiqueta puede ser cualquier identificador legal y es el nombre de la instancia concreta. La palabra clave “component” entre el nombre de la instancia y el nombre del componente es opcional.

El nombre del componente es el nombre declarado al declarar “component”. El nombre del puerto (“port1”) es el nombre del puerto en la especificación del componente. Señal (“signal1”) es el nombre de la señal a la cual se conecta el puerto que se ha especificado antes del símbolo =>. Notemos que se asocian de forma explícita los puertos y las señales mediante la asociación de nombres⁷. Por tanto, aunque el nombre a la derecha e izquierda del símbolo => sea el mismo no existe ambigüedad.

Especificación estructural de un sumador de 4 bits

En la Figura 2.7 se muestra el esquema de circuito del sumador de 4 bits. Las entradas y salidas se especifican como buses de señales cuando es el caso.

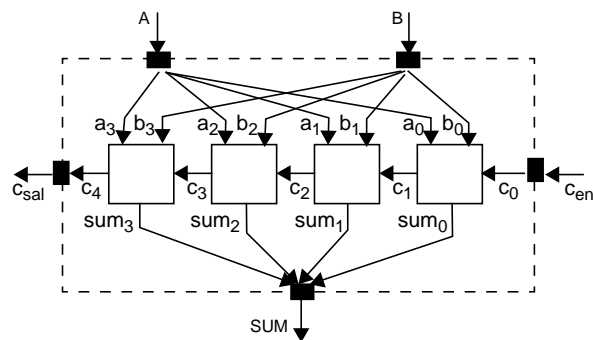


Figura 2.7 Esquema de un sumador de 4 bits construido con sumadores de 1 bit.

La declaración de la interfaz (“entity”) es la misma que en el caso de utilizar un modelo de comportamiento (Figura 2.2).

Después de la declaración de “architecture” y antes de “begin”, hay que declarar los componentes que se utilizan (Figura 2.8).

```
component S1bit
port (x, y, cen : in std_logic; s, csal : out std_logic);
end component;
```

Figura 2.8 Declaración del componente “S1bit”, sumador de un bit.

7. La asociación también puede efectuarse de forma implícita, pero no lo utilizaremos.

La declaración de señales y el resto del cuerpo de la arquitectura se muestra en la Figura 2.9.

```

signal c1, c2, c3, c4: std_logic;
begin
    S1bit0: S1bit port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0));
    S1bit1: S1bit port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1));
    S1bit2: S1bit port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2));
    S1bit3: S1bit port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3));
    csal <= c4;
end structural;

```

Figura 2.9 Instanciación e interconexión de sumadores de un bit (S1bit).

Las señales internas definidas se corresponden con los nombres de la Figura 2.7.

En el Apéndice 2.2 se encuentra el listado completo (instanciación indirecta de entidades mediante componentes).

Modelos parametrizados

Cuando un elemento utiliza parámetros (“generic”) las declaraciones e instanciaciones de los componentes deben reflejar esta situación.

Declaración de un componente (“component”). En la declaración de un componente que ha sido especificado con algún “generic” también se declara el “generic” antes de los puertos.

Sintaxis	Ejemplo
component component_name	component s1bit
generic (identifier_name: type [= value]; ...);	generic (retardoxor: time; retardoand: time; retardoor: time);
[port (port_signal_name: mode type;	port (x: in std_logic;
port_signal_name: mode type;	y: in std_logic;
... port_signal_name: mode type);]	cen: in std_logic;
end component ;	s: out std_logic; csal: out std_logic);
	end component ;

Instanciación de un componente. Una instanciación de un componente declarado con algún “generic” tiene un “generic map” antes del “port map”⁸. Observemos que al final del “generic map” no hay un “;”.

Sintaxis	Comentario	Ejemplo
instance_name: component_name		sum: s1bit generic map (retardoxor => 15 ns,
generic map (identifier_name => value, ...)	Observe que no hay ;	retardoand => 10 ns, retardoor => 15 ns)
port map (port1 => signal1, port2 => signal2, ..., portn => signaln);		port map (x => A(0), y => B(0), cen => cen, csal => c1, s => SUM(0));

8. El valor de un parámetro establecido en un “generic map” al instanciar el componente es prioritario respecto del valor que haya sido establecido en el “generic map” al declarar el componente (“component”).

Utilización de librerías en Quartus

En Quartus los componentes en un diseño jerárquico pueden estar ubicados en directorios. Estos directorios contienen los ficheros que describen las “entity” y “architecture” que se utilizan en una “architecture” más externa del proyecto⁹. A esta posibilidad se la denomina declaración de componentes en la arquitectura¹⁰ o instanciación indirecta de entidades mediante componente (Apéndice 2.2). La idea básica es utilizar el sistema de ficheros para estructurar la organización de los ficheros utilizados en un diseño.

En la página 2 al crear el proyecto, además de incluir los ficheros del diseño, se pueden incluir los directorios donde se almacenan los ficheros que se utilizan como componentes (librerías de usuario en terminología Quartus, Figura 2.10).

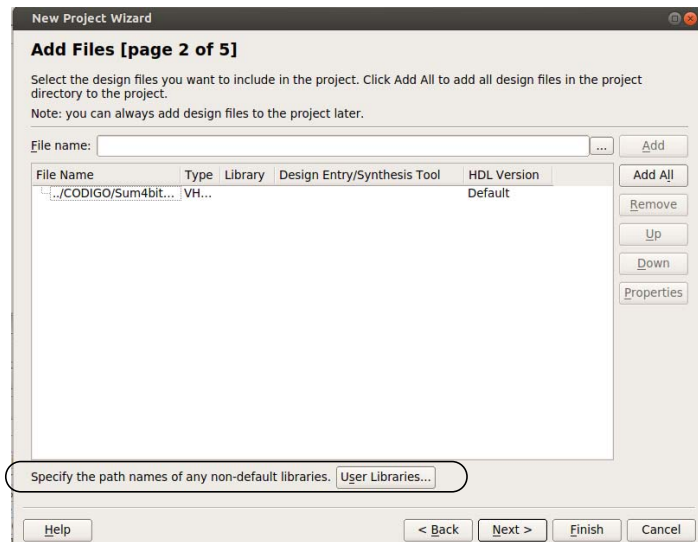


Figura 2.10 Inclusión de librerías de usuario al crear un proyecto en Quartus.

Para incluir los directorios que contienen los ficheros con los componentes (librería de usuario en Quartus) hay que pulsar con el ratón en el rótulo marcado de la Figura 2.10. En la pantalla emergente hay que pulsar con el ratón en el rótulo “...”, asociado a “Project library name”. En la pantalla que emerge hay que posicionarse en el directorio que almacena los ficheros que contienen las “entity” y “architecture” que se utilizan como librería (Figura 2.11)¹¹.

9. Otra posibilidad es la declaración de componentes en librerías (Apéndice 2.2).

10. En Quartus los directorios que contienen componentes se identifican como “Project Library”. Sin embargo, todos los componentes están en la librería “work”.

11. Se pueden utilizar los directorios que se crea necesarios para estructurar el proyecto.

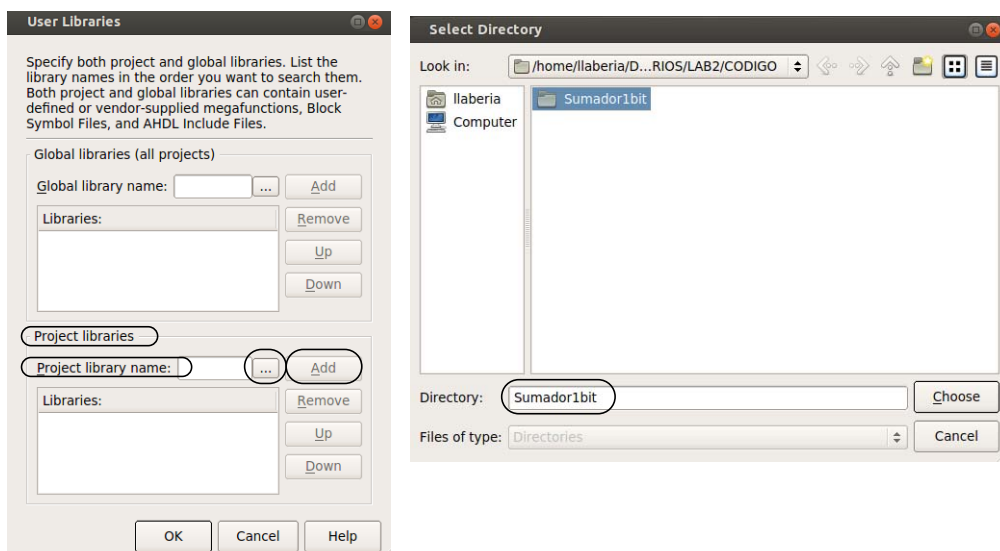


Figura 2.11 Especificación de librerías de usuario.

Una librería de usuario también se puede añadir después de crear el proyecto. Para ello hay que dar la orden “Assignments -> Settings”. La ventana que emerge se muestra en la Figura 2.12. Los pasos que hay que efectuar, para añadir una librería de usuario, son los mismos que han sido descritos previamente (Figura 2.11).

Nota: El nombre de un fichero ubicado en un directorio identificado como librería debe ser idéntico al nombre de la “entity” que contiene, con extensión vhd. Ejemplo, si la “entity” se denomina s1bit, el nombre del fichero debe ser s1bit.vhd.

Trabajo: Para esta práctica ha sido suministrado un fichero que al desempaquetarlo crea una estructura de directorios, incluyendo alguno de ellos ficheros. El directorio raíz es LAB2, el cual incluye el directorio SUMA, que incluye un árbol de directorios.

En el directorio SUMA/COMPONENTES/SUM1BIT/CODIGO está incluida una especificación parametrizada de un sumador de un bit (fichero s1bit.vhd). En el directorio SUMA/SUM4/CODIGO se suministra un fichero con un esqueleto del código de un sumador de 4 bits (S4bits.vhd).

Cree un proyecto Quartus en el directorio SUMA/SUM4/QUARTUS con el nombre S4bits. Este proyecto utiliza el fichero que deberá modificar (S4bits.vhd) y como librería de usuario debe utilizar el directorio SUMA/COMPONENTES/SUM1BIT/CODIGO.

Edite el fichero que especifica de forma estructural y parametrizada (retardos del sumador de 1 bit y número de bits) el sumador de 4 bits. El componente que se utiliza debe declararse en la parte declarativa de la arquitectura.

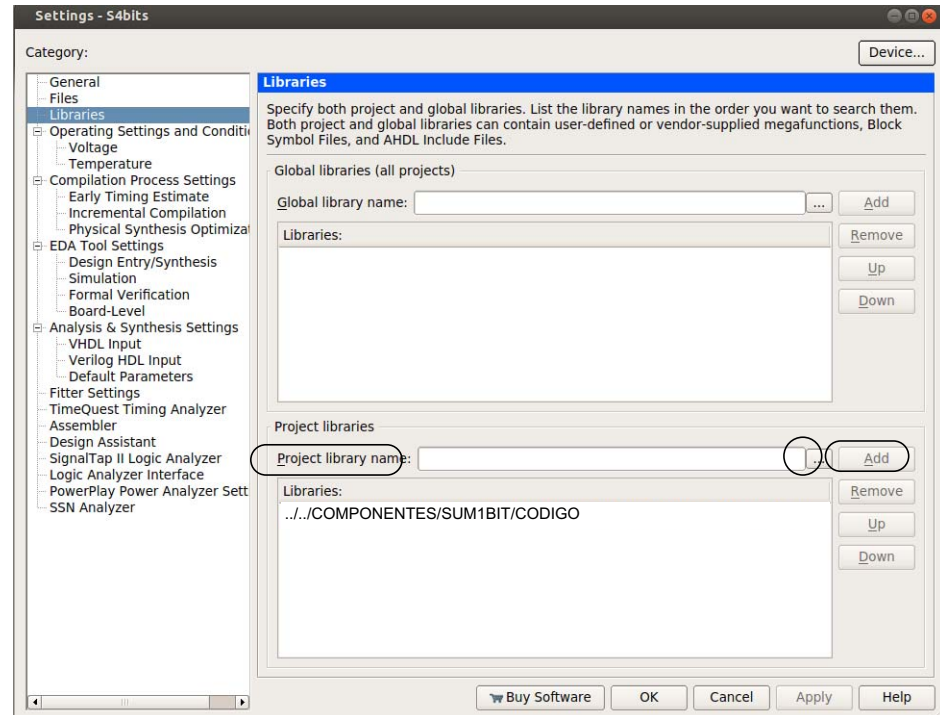


Figura 2.12 Inclusión de librerías de usuario en un proyecto.

Compilación y elaboración RTL

Una vez se han incluido todos los ficheros en el proyecto hay que compilar. Para ello hay que dar la orden "Processing -> Start -> Start Analysis&Elaboration". Después de que la compilación finalice sin errores, en la pestaña "Design Units" de la ventana "Project Navigator", se pueden observar las unidades creadas en el pseudodirectorio work (Figura 2.13). En la pestaña "Hierarchy" se observa la estructura jerárquica del proyecto. Pulsando 2 veces en un elemento, de cualquiera de las pestañas, se abre el fichero correspondiente.

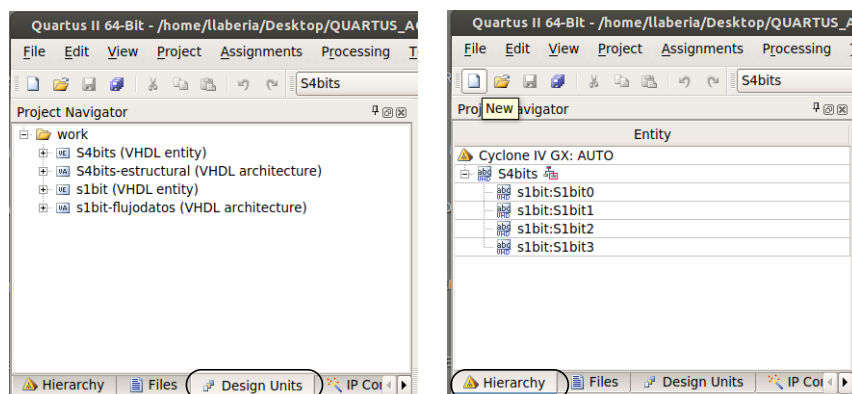


Figura 2.13 Estructura jerárquica del proyecto y unidades creadas.

Análisis de los resultados de compilación. En la ventana “Compilation Report”¹², pestaña “Table of Contents”, se dispone de varios resúmenes de la compilación. En particular, al ser un diseño parametrizado, interesa el resumen “Analysis & Elaboration -> “Parameter Setting by Entity Instance”. Una vez en esta entrada se puede analizar instancia por instancia el valor de los parámetros utilizados.

Elaboración RTL. Quartus efectúa una elaboración RTL (Register Transfer Level) de la descripción del diseño funcional en VHDL¹³. Para ello hay que dar la orden “Tools -> Netlist viewers -> RTL Viewer”. En la pantalla emergente se observa la implementación de la especificación efectuada del sumador de 4 bits y el detalle de un sumador de 1 bit (Figura 2.14). Para observar el contenido de un módulo, junto con todos los otros, hay que seleccionar el módulo, pulsar el botón derecho del ratón y dar la orden “Display Content” (Figura 2.14).

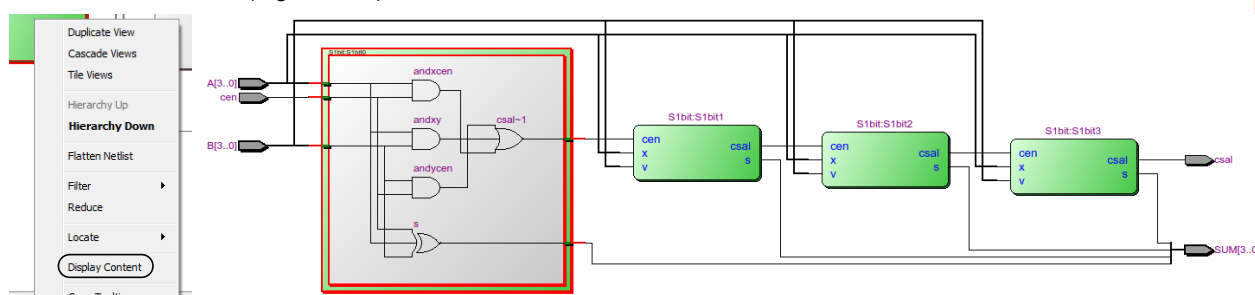


Figura 2.14 Elaboración RTL del diseño del sumador de 4 bits.

12. Si no está activa se puede visualizar dando la orden “Processing -> Compilation Report”.

13. Esta elaboración puede ayudar, por ejemplo, a identificar errores en el conexionado.

Utilizando las pestañas ubicadas en la parte inferior izquierda de la ventana que muestra la elaboración RTL, Quartus muestra, después de seleccionar un elemento, la descripción de los puertos de entrada, salida, etc (Figura 2.15)¹⁴.

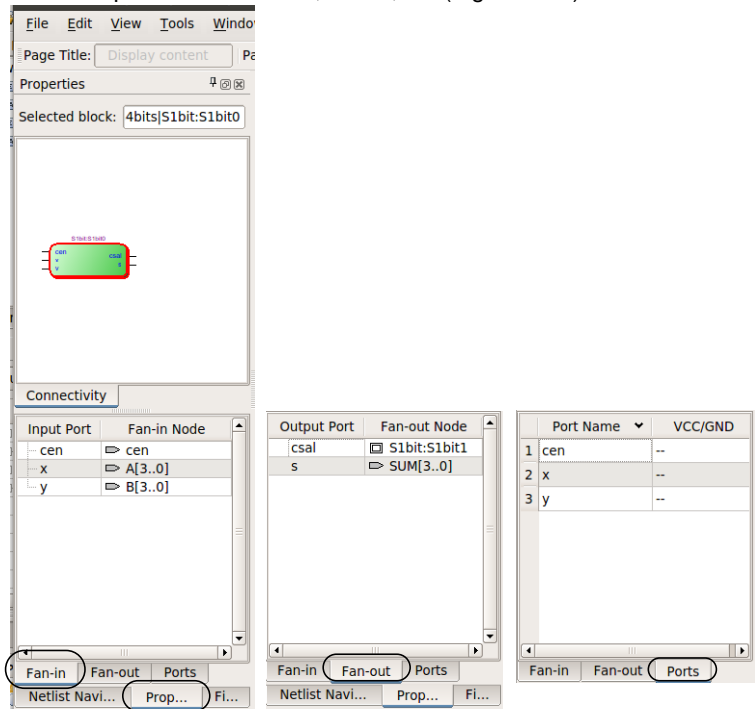


Figura 2.15 Descripción de un elemento en una elaboración RTL del sumador de 4 bits.

Trabajo: Relacione la elaboración RTL con la descripción efectuada en VHDL del sumador de 4 bits.

Simulación

La simulación en Modelsim se activará desde Quartus.

14. Posicionando el cursor en cualquiera de los elementos que constituyen la elaboración RTL Quartus muestra información adicional, después de cierto retardo. También, Quartus muestra información si se selecciona una entrada de un componente.

En la ventana “Library”, de Modelsim, se observan las entidades s1bit y s4bits en el directorio “rtl_work”. Esto es, en el directorio “rtl_work” se incluyen todas las entidades, tanto la descripción estructural como los componentes utilizados (librerías de usuario en Quartus)¹⁵. Para iniciar la simulación en Modelsim pulse dos veces en s4bits (Figura 2.16).

Posteriormente hay que ubicar en la ventana temporal los objetos que se quieran observar. En concreto, se quieren observar los puertos de entrada y salida, A, B, cen, SUM, csal y las señales internas c1, c2, c3 y c4. Para ello, hay que seleccionar en la ventana "Objects" estos objetos y arrastrarlos (seleccionar y sin dejar de pulsar el ratón) a la ventana temporal.

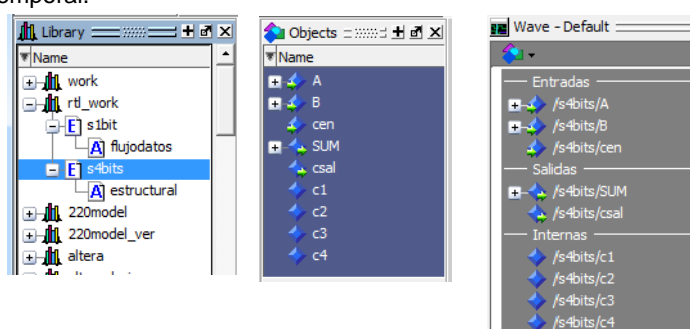


Figura 2.16 *Modelsim. Inicio de la simulación y señales en las ventana de objetos. Ventana temporal después de ubicar las señales y separadores.*

Trabajo: Active la simulación con Modelsim desde Quartus. Inicie la simulación en Modelsim. Incluya en la ventana temporal y textual los puertos de entrada y salida y las señales internas.

Estimulación de las entradas

Las señales de entrada se estimulan con los siguientes valores.

Puertos	Estímulos
A	1010
B	1111
cen	1

Las órdenes en la interface gráfica y las equivalentes en la ventana de mensajes se muestran en la Figura 2.17. En la misma figura se muestra la posibilidad de efectuar la asignación en binario o en hexadecimal (16#).

15. En el Apéndice 2.2 se describe la utilización de librerías distintas a la librerías “work”, denominadas usualmente librerías de recursos.

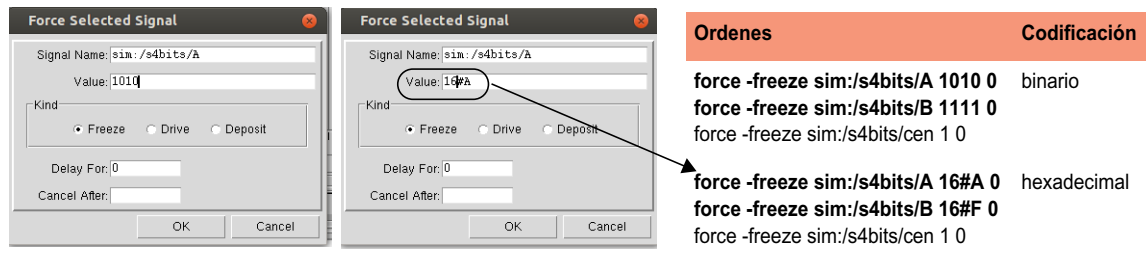


Figura 2.17 Ordenes para inicializar las señales.

Después de simular ("run -all"), la traza de las señales que se observa en la ventana temporal se muestra en la Figura 2.18.

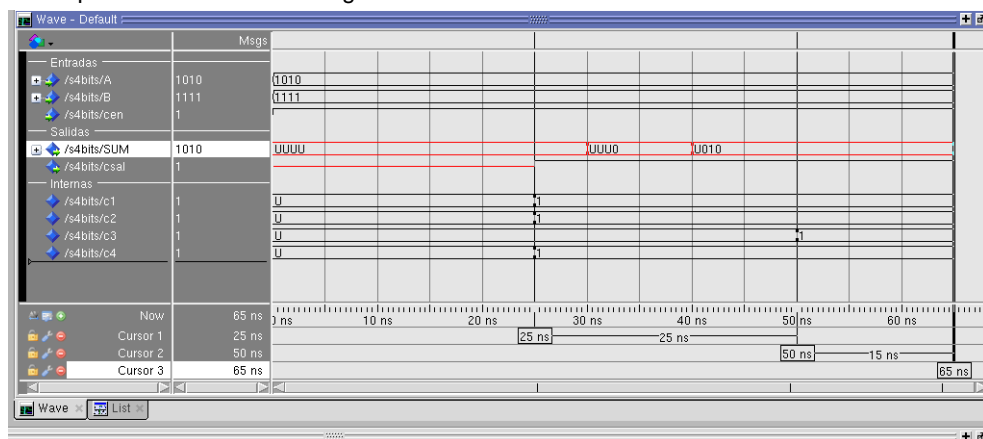


Figura 2.18 Sumador de 4bits. Traza de las señales.

Los acarrees c1, c2 y c4 son estables al cabo de 25 ns (Figura 2.18). Notemos que la suma de los bits correspondientes genera, independientemente del acarreo de entrada, un acarreo. La suma de los bits de peso 2 (ponderación) requiere del acarreo de entrada. La suma de los 3 bits de menor peso es estable a los 40 ns. La suma del bit más significativo requiere del acarreo c3, que no es estable hasta los 50 ns.

La representación de un vector de bits se puede modificar. Por ejemplo, se puede representar en hexadecimal. Para ello, después de seleccionar una señal y pulsar el botón derecho del ratón, hay que dar la orden que se muestra en la Figura 2.19.

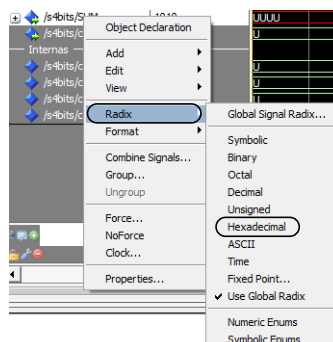


Figura 2.19 Orden para representan un vector de bits en hexadecimal.

En la Figura 2.20 se muestra el diagrama temporal utilizando una representación hexadecimal. Observe que al representar en hexadecimal, la señal SUM se representa como "X" hasta que todos los bits dejan de estar indefinidos.

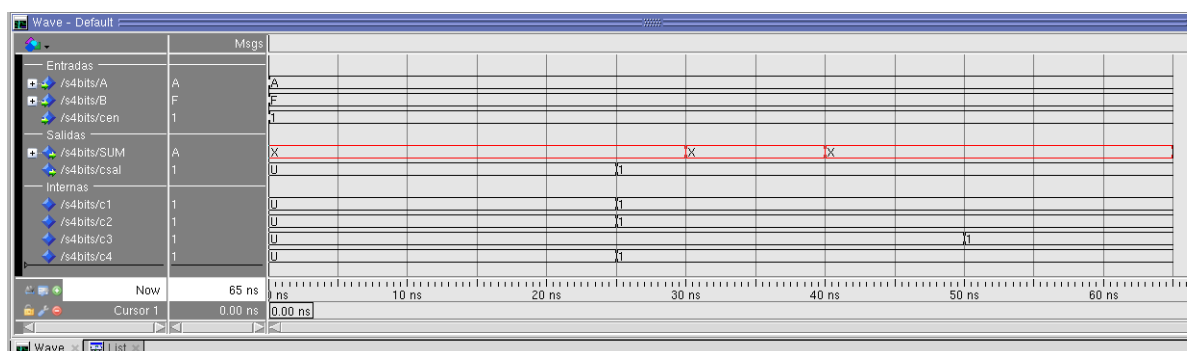


Figura 2.20 Sumador de 4 bits. Traza de las señales en hexadecimal.

También se pueden observar los bits individuales de un vector de bits. Es suficiente pulsar con el ratón en el símbolo (+) que hay a la izquierda del acrónimo que identifica al vector de bits (Figura 2.21).

Registrar señales no visualizadas. En principio, las señales tienen que observarse en la ventana temporal antes de iniciar la simulación. Ahora bien, es factible indicar a Modelsim que registre señales adicionales del diseño, para añadirlas posteriormente a la ventana temporal, sin tener que volver a iniciar la simulación. Para ello, antes de iniciar la simulación, una posibilidad es posicionar el cursor sobre la “entity”, cuyas señales podemos querer observar, en la pestaña “sim”. Seguidamente, al pulsar el botón derecho del ratón emerge una ventana y se da la orden “Add->To Log->All items in design”¹⁶. Por ejemplo, esta alternativa permitirá observar la traza de la señal xorxy de S1bit0 aunque no se ubique en la ventana temporal cuando se efectúa la simulación.

En la Figura 2.23 se muestra la información descrita en la ventana textual.

psql> \setbits/A-q/s4bits/csal-q/s4bits/S1bit0/Q-x-q
/s4bits/B-q/s4bits/S1bit0/cen-q
/s4bits/cen-q/s4bits/C-q/s4bits/S1bit0/s-q
/s4bits/SUM-q/s4bits/c4-q/s4bits/S1bit0/csal-q
/s4bits/c1-q/s4bits/S1bit0/q-q

Q	1010	1111	1	UUUU	U	U	U	U	0	0	1	1	U	U
25000	1010	1111	1	UUUU	1	1	1	1	1	1	1	1	1	1
30000	1010	1111	1	UUUU	1	1	1	1	1	1	0	1	1	0
40000	1010	1111	1	UUUU	1	1	1	1	1	0	1	1	1	0
50000	1010	1111	1	UUUU	1	1	1	1	1	1	0	1	1	0
65000	1010	1111	1	1010	1	1	1	1	1	1	0	1	1	0

Figura 2.23 Salida de la simulación en la ventana textual.

Trabajo: Establezca los valores 1010, 1111, 1 en las señales A, B y cen respectivamente. Posteriormente de la orden "run -all".

Para un análisis un poco más extenso, en el directorio SUMA/SUM4/PRUEBAS dispone del fichero estimulos_retardo.do con la especificación de tres frentes de onda para las señales de entrada¹⁷.

16. También, después de efectuar la selección en la pestaña “sim”, se puede dar la misma orden desde la paleta de órdenes

17. Recuerde la utilización de la orden “Restart” para iniciar una nueva simulación (Figura 1.41 de la Práctica 1). Posteriormente, utilice la ventana textual de Modelsim para dar las órdenes almacenadas en el fichero `estimulos_retardo.do`. Copie las órdenes y posteriormente active la simulación mediante la orden “run-all”. Por otro lado, la ventana textual de Modelsim es una ventana “Linux”. Entonces, una orden para indicar a Modelsim las órdenes relativas a las señales es `do ./../PRUEBAS/estimulos.do`.

Programas de prueba en VHDL (“testbench”)

Un programa de verificación es un programa en VHDL diseñado específicamente para comprobar el funcionamiento de un circuito.

Frente de onda

En una sentencia de asignación de señal la expresión en la parte derecha se denomina elemento de frente de onda. Un elemento de frente de onda describe la asignación de un valor a una señal en un instante de tiempo determinado y tiene la siguiente sintaxis.

Ejemplo	Sintaxis
s <= a and b after 2 ns;	signal_name <= value expression [after time expresion] ;

Después del operador de asignación (<=) distinguimos una expresión que calcula un valor a la izquierda de la palabra clave “after” y una expresión de tiempo a la derecha de la misma palabra clave. La expresión de la izquierda evalúa el nuevo valor que se asigna a la señal y la evaluación de la expresión de tiempo determina el instante de tiempo relativo en el cual la señal tomará el nuevo valor. El valor de la expresión de tiempo se suma al tiempo de simulación actual para determinar el instante de tiempo en el cual la señal recibirá el nuevo valor. Con respecto al tiempo de simulación actual, el par tiempo-valor representa el valor futuro de la señal y se denomina transacción.

En una sentencia de asignación de señal podemos especificar varios elementos de frente de onda y como resultado varias transacciones. Las transacciones (elementos de frente de onda) se especifican en un orden de tiempo creciente. El frente de onda representa la sucesión de valores de una señal (física) en el transcurso del tiempo. Se están describiendo transacciones de una señal en instantes de tiempo futuro. La sintaxis es la siguiente.

Sintaxis
signal_name <= value expr1 [after time expr1], value expr2 [after time expr2], . . . ; siendo expr1 < expr2

En el caso que nos ocupa “value exp” es un objeto constante y en cada transacción se produce un evento ya que el valor de la señal cambia.

Por ejemplo, el frente de onda de la parte superior de la Figura 2.24 se obtiene mediante la sentencia de asignación de señal mostrada en la parte inferior de la misma figura. Notemos que los tiempos son todos relativos al primer elemento del frente de onda.

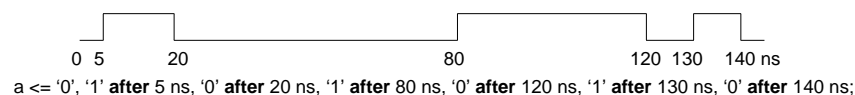


Figura 2.24 Especificación de un frente de onda.

Utilización de Quartus para activar la simulación con un programa de prueba

El programa de prueba no pertenece al diseño. En este sentido, Quartus dispone de un mecanismo para asociar un programa de prueba a un diseño. Este programa de prueba se utiliza en el simulador Modelsim para estimular las señales de entrada del diseño.

Para ello, el programa de prueba además de las señales de estímulo debe instanciar el componente cuyo funcionamiento se quiere comprobar. Se efectúa una descripción estructural. En consecuencia, la “entity” del programa de prueba no tiene puertos (Figura 2.25).

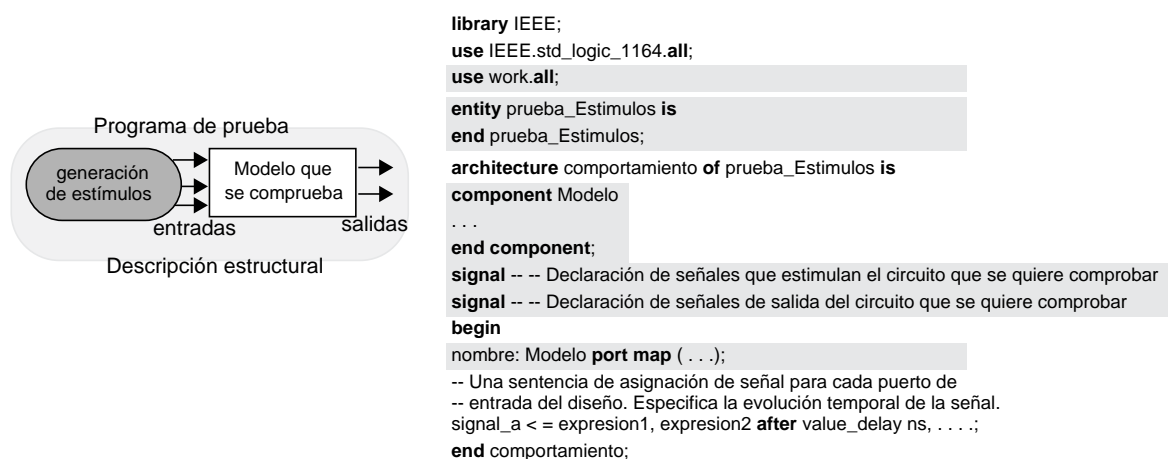


Figura 2.25 Esquema de un programa de prueba en el entorno Quartus.

Seguidamente se muestran posibles frentes de onda para las señales de entrada.

```
a <= "UUUU", "0000" after 200 ns, "UUUU" after 400 ns, "0000" after 600 ns, "UUUU" after 800 ns, "0000" after 1000 ns, "UUUU" after 1200 ns;
b <= "UUUU", "0000" after 200 ns, "UUUU" after 400 ns, "0100" after 600 ns, "UUUU" after 800 ns, "0100" after 1000 ns, "UUUU" after 1200 ns;
cen <= 'U', '0' after 200 ns, 'U' after 400 ns, '0' after 600 ns, 'U' after 800 ns, '1' after 1000 ns, 'U' after 1200 ns;
```

En los frente de onda previos, los valores se especifican en binario. Otra posibilidad es especificar los valores en hexadecimal. Por ejemplo, el frente de onda para la señal b puede especificarse de la siguiente forma.

b <= (others^a =>'U') , x"0" after 200 ns, (others =>'U') after 400 ns, x"4" after 600 ns, (others =>'U') after 800 ns, x"4" after 1000 ns, (others =>'U') after 1200 ns;

a. "Others" es una característica de VHDL que permite definir varios elementos en un vector con el mismo valor.

Trabajo: En el directorio SUMA/SUM4/PRUEBAS se suministra un fichero con el esqueleto del programa de prueba (prueba_s4bits.vhd). Edite el fichero para crear un programa de prueba para el sumador de 4 bits. Como frentes de onda para las entradas puede utilizar los especificados previamente.

Una vez editado el fichero puede abrirlo en el entorno Quartus ("File -> Open"). El fichero no debe incluirse en el proyecto. Puede efectuar un análisis del fichero dando la orden "Processing -> Analyze Current File" (ayuda a corregir algunos errores; no todos).

Una vez se dispone del programa de prueba, hay que indicar a Quartus que la simulación se efectúa con el programa de prueba. Para ello hay que dar la orden "Assignments -> Settings". Emerge una ventana en la que hay que posicionarse en "EDA Tool Settings -> Simulation". En la Figura 2.26 se muestra la ventana en la que Quartus ya ha rellenado algunos campos. Seguidamente hay que pulsar en el círculo que hay a la izquierda de "Compile test bench". Posteriormente hay que pulsar en el rótulo "Test Benches..."

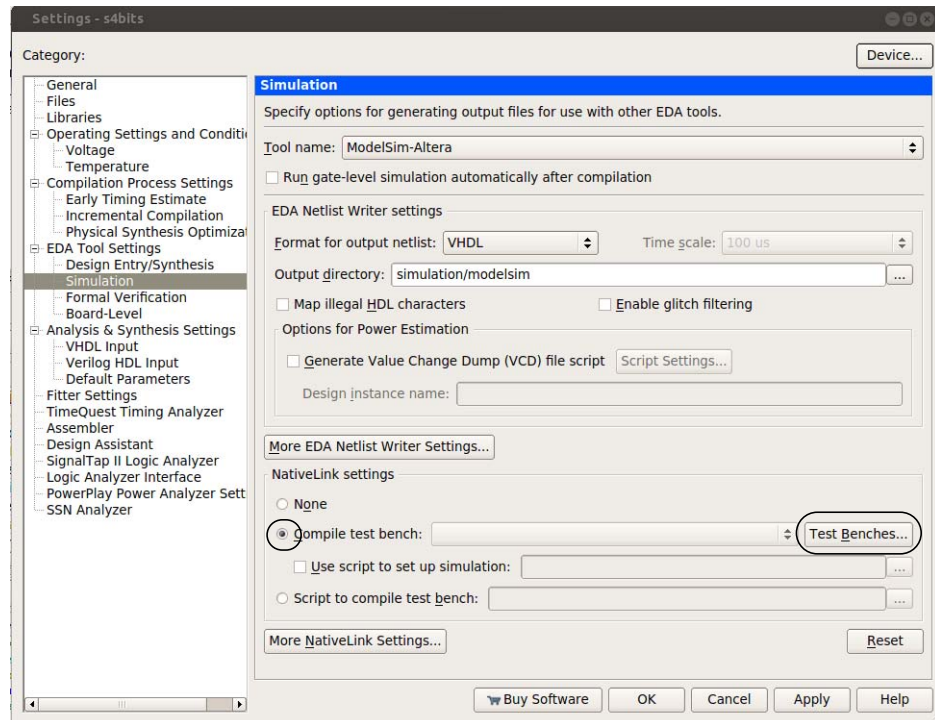


Figura 2.26 Indicación a Quartus de que se quiere utilizar un programa de prueba: 1º paso.

La ventana emergente se muestra en la Figura 2.27. En esta ventana hay que pulsar en el rótulo “New...”. Emerge una nueva ventana que hay que rellenar (Figura 2.27). Los campos que hay que rellenar están marcados en la Figura 2.27. En el campo “Top level module in test bench” hay que indicar el nombre utilizado al declarar “entity” en el programa de prueba.

Una vez rellenado hay que pulsar en el rótulo “...” asociado a “File name”. Emerge una ventana que permite navegar por el sistema de ficheros. El fichero que se utiliza para estimular los puertos de entrada, pruebas4bits.vhd, está en el directorio PRUEBAS. Después de seleccionar el fichero hay que pulsar en el rótulo “Add”. Finalmente hay que ir pulsando “OK” en las distintas ventanas abiertas.

Una vez establecido el programa de prueba que se quiere utilizar, se puede activar Modelsim para efectuar la simulación. Para ello, en Quartus hay que dar la orden “Tools -> Run Simulation Tool -> RTL Simulation”.

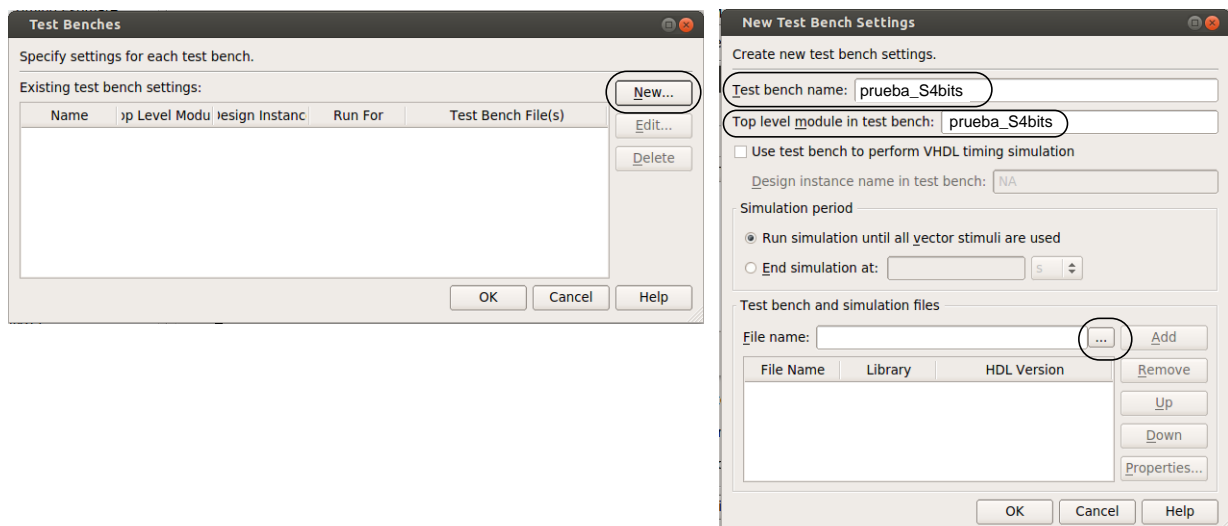


Figura 2.27 Indicación a Quartus de que se quiere utilizar un programa de prueba: 2º paso.

Ficheros creados por QUARTUS. En el fichero QUARTUS/s4bits.qsf se puede observar un conjunto de asignaciones donde se indica que se está utilizando un programa de prueba. En el fichero QUARTUS/simulation/modelsim/s4bits_run_msim_rtl_vhdl.do se puede observar, que además de indicar que se active la simulación (vsim), se indica que se ejecute la simulación (run -all) y que en la ventana temporal se muestren las señales (puertos) de entrada y salida del diseño (do wave.do). En resumen, si no se han producido errores ya se dispone de la simulación¹⁸.

Corrección de errores en el fichero de prueba. Además de la posibilidad comentada en la nota a pie de página, podemos editar directamente el fichero en Modelsim, compilarlo y activar la simulación. Después de modificar el fichero en Modelsim hay que almacenarlo (File -> Save) y compilarlo. Para compilarlo identifique la orden “vcom” que compila el fichero de pruebas en el fichero QUARTUS/simulation/modelsim/s4bits_run_msim_rtl_vhdl.do. Una vez la compilación es correcta, hay que iniciar la simulación. La orden es “vsim ...”, la cual está especificada en el mismo fichero que contiene la orden de compilación. Una vez iniciada la simulación es necesario dar las restantes órdenes que hay en el fichero QUARTUS/simulation/modelsim/s4bits_run_msim_rtl_vhdl.do¹⁹.

Jerarquía de Objetos. La pestaña “sim” muestra una jerarquía de los objetos VHDL (“packages”, componentes, procesos,...) del diseño (Figura 2.28).

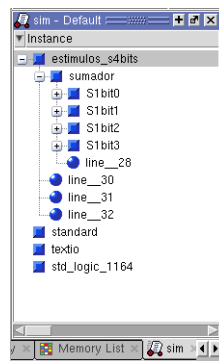


Figura 2.28 Pestaña “sim”. Jerarquía de objetos.

Trabajo: Practique con el diseño del sumador de 4 bits las características y funcionalidades de Modelsim que han sido descritas y las que se describen seguidamente.

18. Si se produce un error al compilar pulse en la línea de la ventana textual que indica el error y Modelsim abre una ventana con el fichero. Analice el fichero y efectúe la modificación necesaria del fichero. Para ello utilice un editor, el cual puede ser el incluido en Quartus. Previamente cierre Modelsim. Una vez efectuada la modificación vuelva a activar la simulación RTL desde Quartus. También puede modificarse en la ventana de edición de Modelsim. Quartus reconoce que ha sido editado y solicita si debe utilizarse la nueva versión.

19. Una alternativa más sencilla, pero que compila todos los ficheros, es dar la orden “do s4bits_run_msim_rtl_vhdl.do” en la ventana de mensajes de Modelsim.

Visualización de señales. Para visualizar de forma amigable la evolución de las señales en la ventana temporal hay que utilizar las funcionalidades de ventana. Por ejemplo, hay que dar la orden "Zoom Full". Esta orden muestra todo el intervalo de simulación en la ventana temporal que se observa en el terminal. (Figura 2.29).

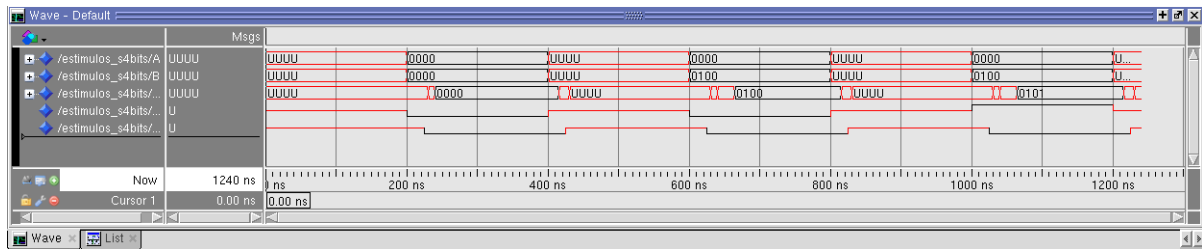


Figura 2.29 Ventana temporal. Valores en binario.

Formateo de las señales. La forma de visualizar la traza de señales en la ventana temporal se puede formatear. Por defecto Modelsim utiliza binario. También es de interés establecer separadores para identificar las señales de entrada y salida del diseño. Además, interesa observar la identificación de las señales (desplazar la barra que separa las dos primeras columnas). En la Figura 2.30 se muestran las señales en formato hexadecimal y en formato literal.

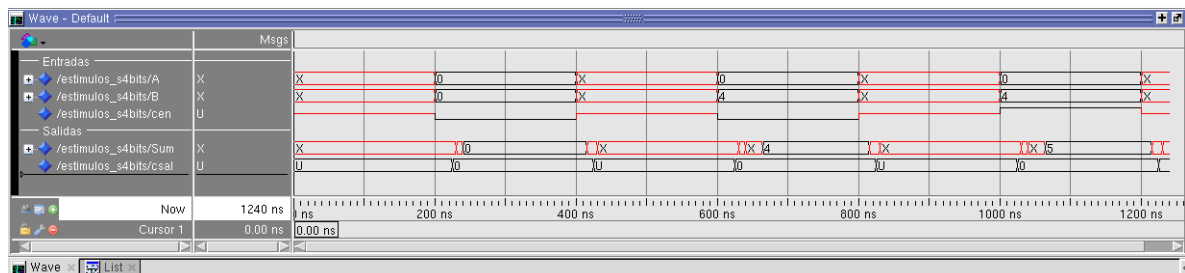


Figura 2.30 Ventana temporal. Valores en hexadecimal y literal.

Visualización de señales internas

Puede ser de interés observar las señales internas del diseño. Para ello hay que seleccionar el identificador "sumador" en la pestaña de la ventana "sim". En la ventana "Objects" se muestran los puertos de entrada y salida del diseño y las señales internas.

Para observar la evolución de las señales internas, es necesario seleccionar las señales c1, c2, c3 y c4 y arrastrarlas a la primera columna de la ventana temporal. En la Figura 2.31 se observa que Modelsim no muestra la evolución temporal. Ello es debido a que, Modelsim por defecto, no almacena la evolución temporal de las señales que no se observan en la ventana temporal.

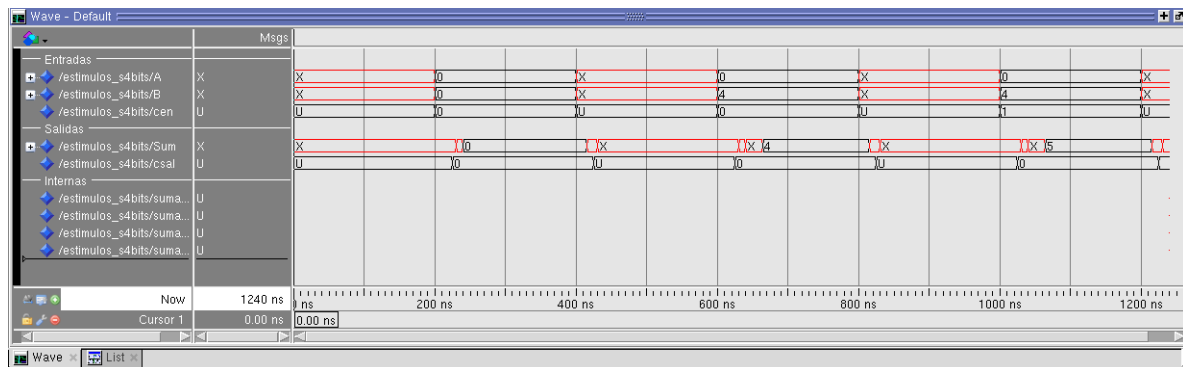


Figura 2.31 Ventana temporal. Señales sin traza temporal debido a que han sido ubicadas en ella después de efectuar la simulación.

Una alternativa para observar las señales añadidas es ubicarlas en la ventana temporal antes de iniciar la simulación. Sin embargo, la simulación ya ha sido efectuada al activar Modelsim desde Quartus y utilizar un programa de prueba. En estas circunstancias Modelsim (indicado por Quartus) sólo muestra los puertos de entrada y salida del diseño que se está comprobando. Por ello es necesario crear un guión ("script") para que Modelsim lo utilice al efectuar la simulación.

Indicación a Quartus del guión que debe utilizar Modelsim al efectuar la simulación. Para ello hay que efectuar los siguientes pasos.

- Almacenar el formato de la ventana temporal. Después de seleccionar la ventana temporal, la orden es "File -> Save Format". El fichero wave.do, o el nombre que se establezca con extensión ".do", se almacena en el directorio PRUEBAS²⁰.
- Crear un fichero en el directorio PRUEBAS con extensión ".do" que contenga las órdenes "do wave.do" y "run -all." Este fichero, el cual ha sido denominado formato_ventanas.do, se suministra en el directorio PRUEBAS del proyecto.
- El siguiente paso es indicar a Quartus que utilice el formato de la ventana temporal que está indicado en wave.do. Para ello, hay que efectuar pasos similares a los que se efectúan para establecer el programa de prueba. En la ventana de la Figura 2.26 pulse en "Use script to set up simulation". Posteriormente pulse en el rótulo "..." asociado. Emerge una ventana para navegar por el sistema de ficheros. Selec-

20. Por defecto, Modelsim propone almacenar el fichero en el directorio "simulation".

ciones el fichero formato_ventanas (Figura 2.32).

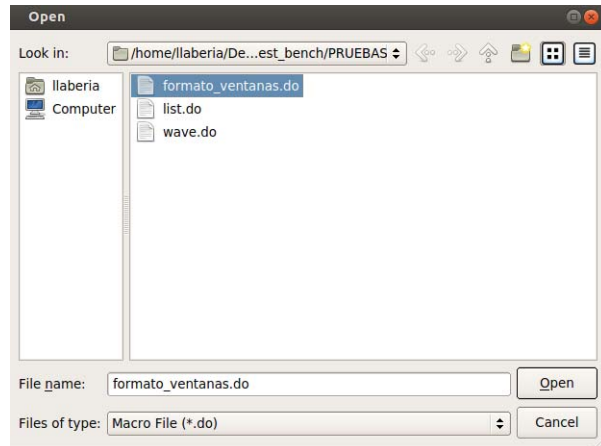


Figura 2.32 Modelsim. Selección del fichero con las órdenes utilizadas para formatear las señales en la ventana temporal.

A partir de ahora, siempre que se active Modelsim desde Quartus, se utilizará el formato de la ventana temporal especificado en el fichero “wave.do”. Puede comprobar este hecho analizando las diferencias entre el fichero

QUARTUS/simulation/modelsim/s4bits_run_msim_rtl_vhdl.do actual y el fichero en la simulación previa (QUARTUS/simulation/modelsim/s4bits_run_msim_rtl_vhdl.do.bak). En la Figura 2.33 se muestra la ventana temporal después de activar la simulación.

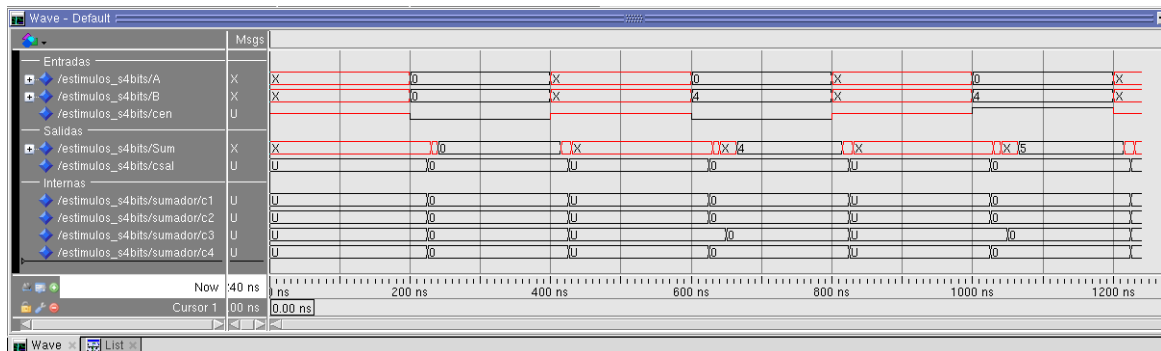


Figura 2.33 Ventana temporal. Inclusión de señales internas.

Almacenamiento por parte de Modelsim de todas las señales. Si ahora nos interesa observar las señales internas en un componente (S1bit) se produce la misma situación que antes. Después de ubicar las señales en la ventana temporal no se observa la evolución temporal. Una solución es añadir en el fichero que se ha creado (formato_ventanas.do), para ser utilizado como guión, la orden “log -r /* ” (sin las

comillas). Con esta orden Modelsim almacena la traza temporal de todas las señales, estén o no en la ventana temporal. En consecuencia, cuando se incluye una señal en la ventana temporal se observa su evolución²¹.

Respecto a la ventana textual los pasos son los mismos. Se debe almacenar el formato en un fichero e incluir la orden “do”, con el nombre del fichero, en el fichero que Modelsim utiliza como guión (script).

En el Apéndice 2.3 se describe una forma de efectuar un análisis del funcionamiento lógico del sumador. En el Apéndice 2.5 se describe cómo observar y medir, utilizando el diagrama temporal, el retardo del circuito para una combinación determinada de las señales de entrada.

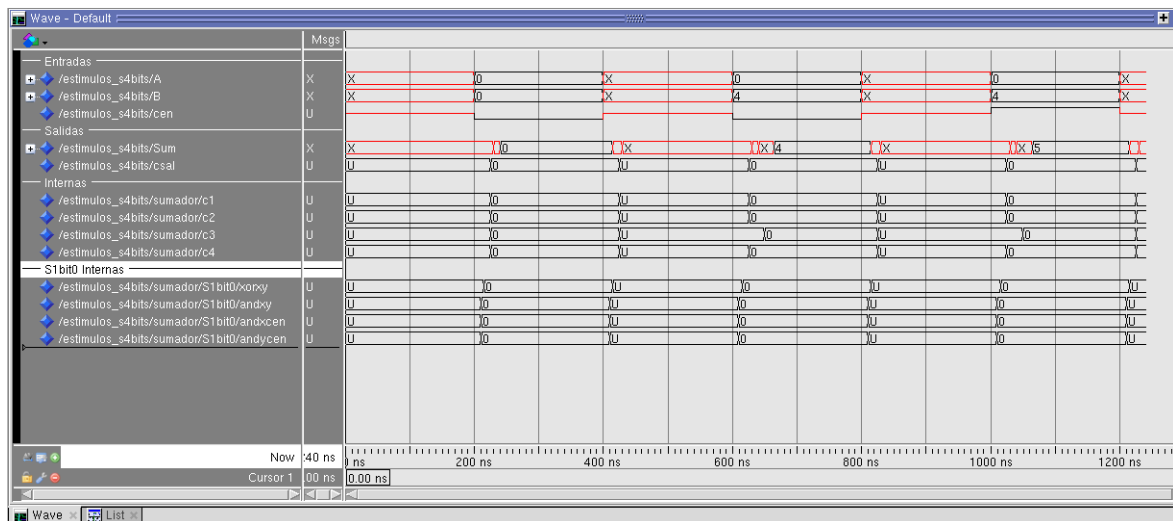


Figura 2.34 Ventana temporal. Inclusión de las señales internas de un sumador de un bit: S1bit0 (peso cero).

Generación de estructuras regulares de componentes en VHDL

Algunos diseños hardware, como el sumador que se ha descrito, se construyen utilizando un mismo elemento varias veces (réplicas). Además, la interconexión de las réplicas sigue un patrón regular. Para facilitar la descripción de estos diseños, VHDL dispone de la sentencia “generate”.

21. Esta orden, debido a la información que debe almacenar, incrementa el tiempo de simulación. Si queremos que estas señales se observen en una nueva simulación sin tener que ubicarlas otra vez, es necesario salvar el formato de la ventana temporal (“File -> Save Format”).

Una sentencia “generate” es una sentencia concurrente que permite especificar la replicación de sentencias concurrentes. Recordemos que una sentencia de instanciación de un componente es una sentencia concurrente.

La etiqueta (label) es necesaria para identificar la estructura. Hay dos formas para generar. La estructura “for” se utiliza para generar de forma iterativa. La estructura condicional (“if”) se utiliza para generar de forma condicional. Después de la palabra clave “generate” se dispone de forma opcional de una parte declarativa, donde se pueden declarar ítems. Estos ítems son los mismos que se pueden declarar en la parte declarativa del cuerpo de una arquitectura. Cada ítem declarado es local a cada replicación de la sentencia concurrente. Seguidamente se especifica la sintaxis.

Sintaxis	Sintaxis	Sintaxis
label: generation_scheme generate [{block_declarative_item } begin] {concurrent_statement} end generate [label];	generation_scheme for generate_parameter_specification if condition	generate_parameter_specification variable in value_u downto value_l variable in value_l to value_u

En la utilización de una sentencia generate hay que:

- identificar réplicas de componentes con una interconexión regular.
- declarar vectores de señales para establecer las conexiones.

El sumador de 4 bits (Figura 2.7), especificado en la Figura 2.9, se puede generar de la forma que se muestra en la Figura 2.35. El parámetro n se ha especificado como un genérico al declarar la “entity” del sumador de 4 bits, de la misma forma han sido declarados los retardos en el sumador de un bit (S1bit).

```

signal c : std_logic_vector (n downto 0);      -- señales para las conexiones
begin
  c(0) <= cen;
  sumador: for i in 0 to n-1 generate
    sumi: s1bit generic map (retardoxor => 15 ns, retardoand => 10 ns, retardoor => 15 ns)
      port map (x => a(i), y => b(i), cen => c(i), s => s(i), csal => c(i+1));
    end generate;
  csal <= c(n);

```

Figura 2.35 Generación de un sumador de n bits a partir de un sumador de 1 bit.

En la Figura 2.36 se muestra un ejemplo de anidación de “for” e “if”.

Ejemplo de anidación

```

label1: for i in range generate
  ...
  label2: if condition generate
    {concurrent_statement}
    end generate [label2];
  end generate [label1];

```

Figura 2.36 Ejemplo de anidación de sentencias “generate”.

Trabajo: En el directorio SUMA/SUMGENERA/CODIGO se suministra un fichero (snbits.vhd) con el esqueleto para generar, utilizando la sentencia “generate”, un sumador de 4 bits a partir de un sumador de 1 bit. Edite el fichero y complete la especificación. En el Apéndice 2.4 se muestran varias posibilidades.

Cree un proyecto Quartus en el directorio SUMA/SUMGENERA/QUARTUS. Este proyecto debe utilizar el sumador de 1 bit (s1bit.vhd) como librería de usuario.

Analice la elaboración RTL y compruebe el funcionamiento. Para esto último copie el programa de prueba, utilizado en SUM4, en el directorio PRUEBAS de este proyecto y modifíquelo de forma oportuna si es necesario²².

-
- Preguntas** **1** Considere el sumador de 4 bits especificado en VHDL mediante sentencias generate. Entregue la elaboración RTL que efectúa Quartus del diseño. Debe visualizarse la especificación RTL de uno de los sumadores.
-

Modelo de comportamiento en VHDL

Las sentencias de asignación de señal permiten especificar fácilmente el comportamiento de puertas lógicas en circuitos digitales. Sin embargo, sistemas digitales de mayor envergadura necesitan descripciones de comportamiento más complejas. Por ejemplo, modelos con lógica combinacional compleja o modelos que utilizan información de estado. Esto es, lógica secuencial (registros, contadores, ...), máquinas de estados.

VHDL permite representar circuitos digitales en diferentes niveles de abstracción, tales como comportamiento, flujo de datos o estructural y en particular RTL. En esta sección se presenta otro constructor para describir el comportamiento de circuitos, en términos de sentencias secuenciales y obtener un modelo que describe de forma abstracta el comportamiento. En particular, en esta sesión de laboratorio, este constructor se utilizará para escribir programas de prueba²³. El elemento base del modelado secuencial o de comportamiento es el constructor “process”.

22. Analice el nombre del componente utilizado y de las señales.

23. En la próxima sesión se utilizará para describir componentes secuenciales.

Proceso (“Process”)

Básicamente un proceso (“process”) es una secuencia de sentencias de asignación que se ejecutan en secuencia, siendo todo el conjunto de sentencias secuenciales una sentencia concurrente. A este conjunto de sentencias de asignación se le denomina cuerpo del proceso.

El cuerpo de un proceso se estructura de forma semejante a un lenguaje de programación clásico. Se declaran y utilizan variables y se dispone de constructores como “if-then”, “if-then-else”, “case”, “for” y “while”. Además un proceso puede contener sentencias de asignación de señal.

Un proceso se ejecuta concurrentemente con otras sentencias de asignación de señal concurrentes. Desde el punto de vista de tiempo de simulación no transcurre tiempo al ejecutarse las sentencias del proceso. Las señales actualizadas toman el nuevo valor al finalizar la ejecución. En este sentido, un proceso es una sentencia de asignación de señal compleja.

Nota: Todos los procesos se ejecutan una vez al iniciarse la simulación.

Recopilando, una sentencia proceso ("process") es el constructor básico, para un modelado que permite utilizar sentencias secuenciales en la especificación del comportamiento de un sistema en función del tiempo. La sintaxis de un proceso es la siguiente:

Sintaxis

```
[process_label:] process [ (sensitivity_list) ] [ is ]
    [ process_declarations ]
    begin
        -- sequential statements
end process [process_label];
```

Un proceso puede tener una etiqueta única con la opción "process_label". La palabra clave "process" sigue a la etiqueta. Después de la palabra clave "process" sigue la lista de activación (sensitivity list"). La lista de activación es un conjunto de señales, separadas por coma, que determinan cuándo el proceso se ejecuta. Un cambio en el valor de una señal en la lista de activación determina que el proceso se ejecute.

Seguidamente se declaran las variables y constantes que se utilizan en el cuerpo del proceso ("process_declarations"), antes de la palabra clave "begin". La palabra clave "begin" indica la parte de inicio de cálculos del proceso (cuerpo). Un proceso finaliza con las palabras clave "end process".

Un proceso se declara dentro del cuerpo de la arquitectura y es una sentencia concurrente. Sin embargo, las sentencias del cuerpo del proceso se ejecutan en secuencia, de forma similar a un lenguaje de programación clásico. Por tanto, el orden de las sentencias es importante, a diferencia del orden de las sentencias de asignación de señales. De

forma semejante a las sentencias concurrentes, un proceso lee y escribe señales, ya sean internas o de la interface. En particular, se pueden efectuar asignaciones a señales que se han declarado externamente al proceso.

Los procesos se comunican mediante señales. Un proceso A puede actualizar una señal que está en la lista de activación de otro proceso B. Esto produce que el proceso B se ejecute después de que el proceso A haya actualizado la señal. A su vez el proceso B puede actualizar una señal que esta en la lista de activación del proceso A.

Objetos de datos en VHDL: variables

La sintaxis para la declaración de un objeto variable es la siguiente.

Ejemplos	Sintaxis
variable var: std_logic ; variable index: integer range (0 to 255) := 0; variable index2: integer := -34;	variable variable_name: type := initial value;

Notemos la similitud con las declaraciones de los tipos “signal” y “constant”. En el segundo ejemplo se indica un rango de enteros. Las variables se declaran dentro de un proceso y no son visibles fuera del proceso.

Operador de asignación de variables. El operador de asignación de variables es “:=”. Notemos que es el mismo operador que se utiliza para inicializar señales, variables y constantes.

Sentencias secuenciales (“sequential statements”)

El término sentencia secuencial es debido al hecho de que las sentencias en el cuerpo de un proceso se ejecutan en secuencia. La ejecución de las sentencias secuenciales se inicia cuando se produce un cambio en una señal contenida en la lista de activación. La ejecución de las sentencias dentro de un proceso continua hasta la última sentencia. Después de la ejecución de la última sentencia, el control vuelve al inicio del proceso.

Sentencia de asignación de señal. Un tipo de sentencia secuencial es la sentencia de asignación de señal que hemos estado utilizando hasta ahora. Ahora bien, esta sentencia se ejecuta en el orden secuencial en que está escrita. Recordemos que un proceso sólo se ejecuta si se modifica una señal en la lista de activación. Por tanto, la sentencia de asignación de señal sólo se ejecutará en este caso.

Dada una sentencia de asignación de señal en el cuerpo de un proceso, el simulador únicamente modifica el valor de la señal cuando el proceso se suspende. En el caso de existir una cláusula “after” en la sentencia de asignación de señal, la modificación del valor se efectúa después del retardo que ha sido especificado, partiendo del tiempo actual.

En resumen, el valor de una señal es actualizado únicamente después de la ejecución de todas las sentencias del cuerpo del proceso o cuando se encuentra una sentencia “wait”²⁴. Una consecuencia es que si al ejecutar el cuerpo del proceso se efectúan varias asignaciones a la misma señal, sólo es efectiva la última asignación. Otra consecuencia es que en cualquier lectura de una señal, en el cuerpo del proceso en el cual se actualiza el valor de la señal, se lee el valor establecido en la anterior suspensión²⁵.

Seguidamente describiremos otros tipos de sentencias secuenciales.

Sentencia de asignación de variable. Las sentencias de asignación de variables en un proceso se ejecutan inmediatamente y la acción de asignación se indica mediante el operador “:=”. Este comportamiento es distinto del comportamiento en una sentencia de asignación de señal (“<=”) donde el cambio se produce después del retardo especificado²⁶. Por tanto, los cambios que se efectúan en una variable son visibles (están disponibles) en sentencias posteriores del mismo proceso y se pueden utilizar para almacenar valores intermedios.

Una variable mantiene su valor de una ejecución del proceso a la siguiente. Si una variable se lee antes de que se asigne un nuevo valor se comporta como una posición de almacenamiento.

Sentencia “if”. En una sentencia “if” la secuencia de sentencias que se ejecuta depende de una o más condiciones. La sintaxis es la siguiente.

Sintaxis
<pre> if [() condition []] then sequential statements [elsif condition then sequential statements] [else sequential statements] end if;</pre>

Cada sentencia “if” tiene asociada la palabra clave “then”. Cada sentencia “if” finaliza con un “end if”. Si se quiere utilizar un constructor “else if” la versión VHDL es “elsif”. La cláusula final “else” no tiene asociada con ella la palabra clave “then”. La cláusula “else” se ejecuta cuando no se cumple ninguna de las condiciones previas. En estas condiciones la sentencia “if” garantiza que al menos una de las secuencias de sentencias se

24. Posteriormente se describe la utilización de la sentencia “wait” en un proceso. En particular, una sentencia “wait” suspende la ejecución del proceso.

25. Los resultados observados en las simulaciones del código VHDL especificado por el usuario y el código VHDL generado por la herramienta de síntesis pueden ser distintas. Las herramientas de síntesis no infieren posiciones de almacenamiento cuando sólo existe una posible secuencia de sentencias en el cuerpo del bucle. En la siguiente sesión se muestra un ejemplo en un apéndice, ya que en este curso no nos centramos en la síntesis.

26. Por defecto hay un retardo delta mayor que cero, que es efectivo al suspender el proceso. Posteriormente se expone con mayor detalle la diferencia entre una señal y una variable.

ejecuta. La cláusula final “else” es opcional. Su no inclusión introduce la posibilidad de que no se ejecute ninguna de las secuencias de sentencias asociadas con las condiciones evaluadas²⁷.

Cada condición es una expresión lógica (booleana). La sentencia “if” comprueba cada condición en el orden en el cual han sido escritas, hasta que se encuentra una condición que se cumpla.

Las sentencias “if” pueden imbricarse.

Tipos de datos en VHDL

El resultado de una comparación ($s = '0'$) es un valor boolean y se utiliza en sentencias condicionales. El lenguaje VHDL tiene predefinido el tipo boolean.

Tipo boolean. Se distinguen los valores: true y false.

Los valores “true” y “false” de tipo boolean no son equivalentes (o intercambiables) con los valores ‘1’ y ‘0’ respectivamente de std_logic. Seguidamente se muestran dos ejemplos. El ejemplo que se muestra a la izquierda es incorrecto. Debemos efectuar la especificación de la forma que se indica a la derecha. El objeto d en el ejemplo es de tipo std_logic.

Ejemplo incorrecto	Ejemplo correcto
if d then	if d = '1' then

Señales versus variables

Podemos decir que una señal representa un cable o algún tipo de conexión física en un diseño. Una señal toma valores en instantes específicos de tiempo en la simulación. Una variable toma el valor inmediatamente cuando es asignada. Seguidamente se utilizan dos ejemplos para mostrar la diferencia entre la asignación de variables y señales en el cuerpo de un proceso (Figura 2.37).

27. Esta posibilidad es utilizada para especificar posiciones de almacenamiento. La herramienta de síntesis inferirá tales posiciones de almacenamiento. En la próxima sesión se detalla su utilización.

En el ejemplo de la izquierda de la Figura 2.37 el objeto `v_a` se declara como una variable. Entonces, la sentencia `S2` utiliza el valor establecido en `v_a` en la sentencia `S1`.

Ejemplo: variables	Ejemplo: señales
<pre>p_var: process (c, d) is variable v_a: std_logic; begin S1: v_a := c and b; S2: ... := v_a ... S3: ... <= v_a ... end process p_var;</pre>	<pre>signal s_a: std_logic; p_señ: process (c, d) is begin S1: s_a <= c and b; S2: ... := s_a ... S3: ... <= s_a ... end process p_señ;</pre>

Figura 2.37 Ejemplos de asignación de variables y señales.

En el ejemplo de la derecha de la Figura 2.37 el objeto `s_a` se declara como una señal. La sentencia `S2` no utiliza el valor establecido en la sentencia `S1`. Utiliza el valor que la sentencia `S1` ha establecido en la ejecución previa del proceso. Ello es debido a que una señal toma el valor asignado después de un retardo. Por defecto, este retardo es un valor denominado *delta* que es mayor que cero. Por tanto, como un proceso se ejecuta en tiempo cero, el valor asignado a una señal no es observable en una sentencia que se especifica posteriormente en el cuerpo del proceso. En otras palabras, se ignoran dependencias textuales entre señales. En la Figura 2.38 se ilustra de forma esquemática el proceso de actualización de señales en un proceso.

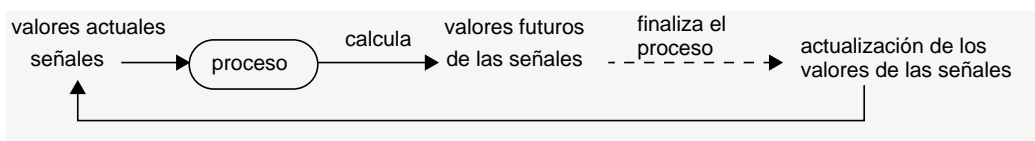


Figura 2.38 Actualización de señales en un proceso.

Módulo de estimulación y verificación

El constructor `proceso` se puede utilizar para describir en VHDL la generación de señales de estimulación de un circuito. En particular se puede utilizar para generar una señal de reloj y adicionalmente delimitar el número de ciclos. También se puede utilizar para comprobar el funcionamiento de un circuito. Antes de mostrar la especificación introduciremos otras funcionalidades disponibles en VHDL.

En este contexto de estimulación y comprobación, se puede explotar toda la expresividad del lenguaje VHDL, ya que usualmente el programa no se sintetiza en hardware.

Sentencia secuencial `wait`. En un proceso puede no especificarse una lista de activación. Ahora bien, en este caso es necesario que el proceso tenga alguna sentencia con la palabra clave `wait`. La sentencia `wait` suspende la ejecución de un proceso hasta que se produce un evento. Algunas de las posibilidades son:

Ejemplos	Sintaxis
wait for 10 ns; wait until x = '1';	wait for time expression; wait until condition; wait ;

La condición en la sentencia “wait until” debe cumplirse (“true”) para que el proceso reanude la ejecución. En el primer ejemplo, el proceso se espera hasta que hayan transcurrido 10 ns. Si sólo se especifica la sentencia wait el proceso queda suspendido indefinidamente.

En un proceso, si se especifica una lista de activación no se puede especificar una sentencia con la palabra clave “wait” y viceversa.

En la Figura 2.39 se muestra, como ejemplo de utilización de la sentencia wait, la generación de una señal de reloj cuadrada con un periodo de 10 ns. Se generan seis periodos de reloj. La sentencia “wait” suspende indefinidamente el proceso cuando han sido generados los 6 periodos.

Señal de reloj	
library ieee; use ieee.std_logic_1164.all; use IEEE.numeric_std.all; entity reloj is port (reloj : out std_logic); end reloj;	architecture compor of reloj is begin reloj: process variable iter : integer := 0; begin reloj <= '1'; wait for 5 ns; reloj <= '0'; wait for 5 ns; iter := iter +1; if (iter = 6) then wait ; end if ; end process ; end compor;

Figura 2.39 Especificación en VHDL de una señal de reloj cuadrada con un periodo de 10 ns durante 6 periodos.

Sentencia “for loop”. La sentencia “for loop” es una sentencia secuencial que se utiliza para ejecutar un conjunto de sentencia secuenciales de forma repetida. El parámetro del “for loop” va tomando cada uno de los valores especificados en el rango de izquierda a derecha. El parámetro de la sentencia “for loop” está implícitamente declarado y solamente es visible en el cuerpo del bucle. Los atributos de un objeto también se pueden utilizar para especificar un rango. La sintaxis es la siguiente.

Sintaxis	Ejemplo
<pre>[Label:] for parameter_name in range loop sequential statements . . . end loop [Label];</pre>	<pre>signal a: std_logic_vector (3 downto 0); asig: process (a) variable tmp: std_logic; begin tmp := '0'; for i in 0 to 3 loop tmp := tmp or a(i); end loop; b <= tmp; end process;</pre>

La especificación del rango puede ser de la forma “downto” o “to”. También se puede especificar utilizando el atributo “range”.

Atributos VHDL. Los atributos se utilizan para obtener información sobre señales y variables (ítem). Los atributos predefinidos en VHDL se aplican a un prefijo tal como una señal o una variable. Hay varias clases de atributos. Por ahora nos centraremos en los aplicables a objetos “scalar” y “array”. En el Apéndice 2.4 se detalla la sintaxis y hay más ejemplos.

Seguidamente se relacionan algunos atributos útiles al especificar el constructor “for loop”.

Atributos	Sintaxis	Ejemplo
t: cualquier objeto	range upper downto lower lower to upper	<pre>variable a: integer range 0 to 3 for i in 3 downto 0 loop for i in 0 to 3 loop</pre>
t'high: el mayor valor de t t'low: el menor valor de t	t'high downto t'low t'low to t'high	<pre>for i in a'high downto a'low loop for i in a'low to a'high loop</pre>
A: objeto tipo array A'range: es el rango A'left to A'high o el rango A'high downto A'left	A'range	<pre>signal a: std_logic_vector (3 downto 0); for i in a'range loop</pre> a'range: (3 downto 0)
A'left: es el valor más a la izquierda de tipo t (mayor si downto)	A'left	a'left: 3
A'high: es el valor más a la derecha de tipo t (menor si to)	A'high	a'high: 0

Sentencia “while loop”. La sentencia “while loop” es una sentencia secuencial que se utiliza para ejecutar un conjunto de sentencias secuenciales de forma repetida condicionalmente. En cada iteración se comprueba una condición. El bucle finaliza cuando no se cumple la condición.

Sintaxis	Ejemplo
<pre>[Label:] while (condition) loop sequential statements . . . end loop [Label];</pre>	<pre>signal a: std_logic_vector (3 downto 0); asig: process (a) variable tmp: std_logic; variable i: integer; begin tmp := '0'; i := 0; while i < 4 loop tmp := tmp or a(i); end loop; b <= tmp; end process;</pre>

Una vez descritos los constructores para especificar de forma repetida la ejecución de sentencias nos centramos en la construcción de programas de prueba.

En los siguientes apartados se muestran diferentes estructuras de un programa de prueba. En un programa de prueba se pueden identificar varias partes:

- Generación de los datos para estimular las entradas del dispositivo que se comprueba ("Device Under Test, DUT").
- Generación de la señal de reloj para la el DUT, si es necesaria.
- Generación de señales de inicialización, si son necesarias.
- Verificación de los resultados del DUT para un conjunto de datos de entrada.

Generación de los estímulos

Los frentes de onda utilizados para la comprobación del funcionamiento del sumador de 4 bits se pueden modelar mediante el constructor "process".

En un frente de onda los valores de tiempo que se especifican para estimular las entradas representan lo que podemos denominar tiempo absoluto. Los tiempos que se indican son respecto al instante de inicio de la simulación. El frente de onda se especifica mediante una sentencia de asignación concurrente (Figura 2.40).

```
a <= x"0" after 100 ns, x"0" after 300 ns, x"0" after 500 ns, x"0" after 700 ns, x"2" after 900 ns, x"5" after 1100 ns, x"1" after 1300 ns;  
b <= x"0" after 100 ns, x"4" after 300 ns, x"4" after 500 ns, x"A" after 700 ns, x"A" after 900 ns, x"2" after 1100 ns, x"1" after 1300 ns;  
cen <= '0' after 100 ns, '0' after 300 ns, '1' after 500 ns, '1' after 700 ns, '1' after 900 ns, '1' after 1100 ns, '1' after 1300 ns;
```

Figura 2.40 Frentes de onda para estimular un sumador de 4 bits.

Generación de señales de estímulo. En este apartado se muestra la generación de señales de estímulo utilizando tiempos que podemos denominar relativos. El instante de asignación de valores a las señales de entrada se indica de forma relativa al instante de asignación previo. En la Figura 2.41 se muestra la generación de los frentes de onda de la Figura 2.40 mediante el constructor "process". Entre conjuntos de estímulos se utiliza una sentencia "wait" para esperar como mínimo el retardo del sumador. El resultado se comprueba analizando el diagrama temporal. La última sentencia "wait" se utiliza para suspender la simulación de forma indefinida.

```

Generación de señales de estímulo
architecture comp of prueba is
begin
prueba: process
begin
a <= x "0";
b <= x "0";
cen <= '0';
wait for 100 ns;
a <= x "0";
b <= x "4";
cen <= '0';
wait for 200 ns;
a <= x "0";
b <= x "4";
cen <= '1';
wait for 200 ns;
...
wait;
end process;
end comp;

```

Figura 2.41 Generación de señales de estímulo mediante el constructor “process”.

Preguntas

La siguiente pregunta se refiere al sumador de 4 bits con propagación serie del acarreo descrito (“Esquema de un circuito sumador de 4 bits” on page 57).

- 2 Suponga que antes del instante de tiempo t_0 todas las señales del sumador son estables y que en este instante solo cambia una las 9 señales de entrada. Suponga también que la última señal de salida que se estabiliza es s_2 en el instante t_1 . Indique el retardo del circuito (caso peor), los valores de las entradas antes del instante t_0 y en el instante t_0 . Modifique el programa de prueba (prueba_S4bits.vhd)²⁸ añadiendo un proceso que genere las señales de entrada correspondientes (“Generación de señales de estímulo mediante el constructor “process”.” on page 93). Considere $t_0=200$ ns. Compruebe el resultado con el simulador. Tenga en cuenta que los parámetros asociados a los retardos de las puertas se establecen al instanciar el componente S4bits en el programa de prueba (“Modelos parametrizados” on page 63).

28. Comente las sentencias de asignación de señal utilizadas en un trabajo previo.

Comprobación de los resultados

Además de generar los estímulos el proceso puede utilizarse para comprobar el resultado de forma automática²⁹. El proceso, después de inyectar las señales de estímulo, espera un retardo en consonancia con el retardo del diseño y comprueba las salidas del dispositivo. Estas acciones se repiten para cada conjunto de señales de estímulo.

Para efectuar la comprobación se utiliza la sentencia “assert”. Cuando no se cumple la condición especificada en la sentencia “assert” se imprime un mensaje. La sintaxis de la sentencia “assert” es la siguiente.

Sintaxis	Ejemplo
assert condition [report expression] [severity expression]	assert csal = acarreo_salida report “error en el acarreo de salida” severity error;

Después de la palabra clave “assert” se especifica la condición que debe cumplirse. Las palabras clave “report”³⁰ y “severity” son opcionales.

Cuando no se cumple la condición se imprime el texto que se indica en “expression” asociada a la palabra clave “report”. Si ésta ha sido omitida se imprime el mensaje “Assertion violation”.

La palabra clave “severity” se utiliza para indicar un nivel de severidad en el error detectado y las acciones que toma el simulador. Los niveles han sido declarados como

Declaración
type severity_level is (note, warning, error, failure) ;

Si se omite la palabra clave “severity” el valor por defecto es “error”.

La palabra clave “report” acepta una única cadena de caracteres, la cual puede construirse a partir de subcadenas de caracteres. Para que una cadena de caracteres se imprima en varias líneas es necesario utilizar los caracteres especiales CR (retorno de carro) y LF (nueva línea) al concatenar subcadenas de caracteres.

29. Esta posibilidad elimina la necesidad de inspeccionar la traza temporal para comprobar el resultado, lo cual es tedioso.

30. La sentencia report se puede utilizar independientemente de la sentencia assert.

Por otro lado es de interés imprimir, por ejemplo, el valor de las señales de los estímulos cuando no se cumple la condición. Para ello se utiliza el atributo “image”, que permite producir una cadena de caracteres que representa un objeto escalar (std_logic o integer)³¹. En la Figura 2.42 se muestran dos ejemplos.

Ejemplo

```
assert csal = '1'
report "error en el acarreo de salida" & CR & LF &
"acarreo de salida: " std_logic'image(csal) & CR & LF &
"acarreo esperado: 1"
severity error;
```

Ejemplo

```
assert suma = x"4" and csal = '1'
report "error en el resultado" & CR & LF &
"estímulo a: " & integer'image(to_integer(unsigned(a))) & " b: " & integer'image(to_integer(unsigned(b))) & "cen: " & std_logic'image(cen)
& CR & LF &
"resultado del diseño. Acarreo: " & std_logic'image(csal) & "suma: " & integer'image(to_integer(unsigned(suma))) & CR & LF &
"resultado esperado. Acarreo: 1" & "suma: 4"
severity error;
```

Figura 2.42 Ejemplos de comprobación de resultados e impresión textual de los mismos en caso de error.

Comprobación automática. Los estímulos y el resultado, para efectuar la comprobación, pueden generarse en el propio programa de prueba. Para ello se utilizan bucles que permiten generar combinaciones de los valores de estímulo de forma metódica y generar el resultado que producen estos valores de estimulación.

En el caso del diseño que nos ocupa, un sumador de vectores de bits, los estímulos se pueden generar mediante la función de conversión de entero a “std_logic_vector” de la variable de iteración del bucle. El resultado se puede calcular utilizando aritmética entera y convirtiendo el valor a “std_logic_vector”³² (Figura 2.42).

31. Para mostrar los valores en hexadecimal o binario es necesario utilizar funciones que conviertan el tipo `std_logic_vector` a una cadena de caracteres. Estas funciones están disponibles en VHDL 2008.

32. También se puede convertir el resultado del sumador, que se está comprobando, a entero, mediante la función correspondiente.

Generación de señales de estímulo y comprobación del resultado

```

architecture comp of prueba is
begin
prueba: process
begin
cen <= '0';
for i in 0 to 3 loop
for j in 0 to 3 loop
a <= std_logic_vector(to_unsigned(i,4));
b <= std_logic_vector(to_unsigned(j,4));
wait for periodo;
assert (to_integer(unsigned(csal & suma)) = (i+j))
report "la comprobación falla" severity error;
end loop;
end loop;
cen <= '1';
...
wait;
end process;
end comp;

```

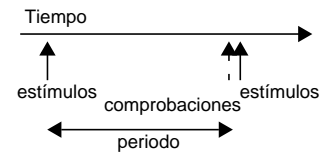


Figura 2.43 Programa de prueba. Generación de estímulos y comprobación del resultado.

Comprobación mediante un modelo de referencia. Un diseño concreto se puede comprobar utilizando una especificación funcional del diseño, efectuada en VHDL. Esta especificación funcional ha sido comprobada previamente de forma exhaustiva y se utiliza como modelo de referencia para otros diseños.

En la Figura 2.44 se muestra una especificación del sumador, efectuándose los cálculos en aritmética entera.

Modelo de referencia

<pre> library IEEE; use IEEE.std_logic_1164.all; use IEEE.numeric_std.all; entity Snbitsref is generic (tam: positive); port (A: in std_logic_vector (tam - 1 downto 0) ; B: in std_logic_vector (tam - 1 downto 0) ; cen: in std_logic; SUM: out std_logic_vector (tam - 1 downto 0) ; csal: out std_logic); end Snbitsref; </pre>	<pre> architecture compo of Snbitsref is signal suma: integer; signal cociente, t_cen: std_logic_vector (1 downto 0); begin t_cen <= '0'&cen suma <= to_integer(unsigned(A)) + to_integer(unsigned(B)) + to_integer(unsigned(t_cen)) ; SUM <= std_logic_vector(to_unsigned((suma mod 2**(tam)),tam)); cociente <= std_logic_vector(to_unsigned((suma/(2**(tam))),2)); csal <= cociente(0); end compo; </pre>
--	--

Figura 2.44 Sumador donde los cálculos se efectúan en aritmética entera.

Trabajo: En el directorio SUMA/SUMGENERA/PRUEBAS se suministra un fichero que permite efectuar la comprobación de forma automática utilizando un modelo de referencia (prueba_snbits.vhd). Para ello, también se suministra un fichero con el modelo referencia (snref.vhd),

Utilización de una señal de reloj. Para especificar los instantes de inyección de estímulos y comprobación del resultado se puede utilizar una señal de reloj. El esquema típico es utilizar un reloj cuyo periodo es el doble del retardo del circuito combinacional. En el flanco ascendente se inyectan estímulos y en el flanco descendente se comprueba el resultado³⁴.

El reloj se genera mediante un proceso como el mostrado en la parte izquierda de la Figura 2.46. La estructura del proceso que genera los estímulos se muestra en la parte derecha de la Figura 2.46³⁵. Para parar el reloj, cuando han finalizado los estímulos, se utiliza una variable compartida o una señal.

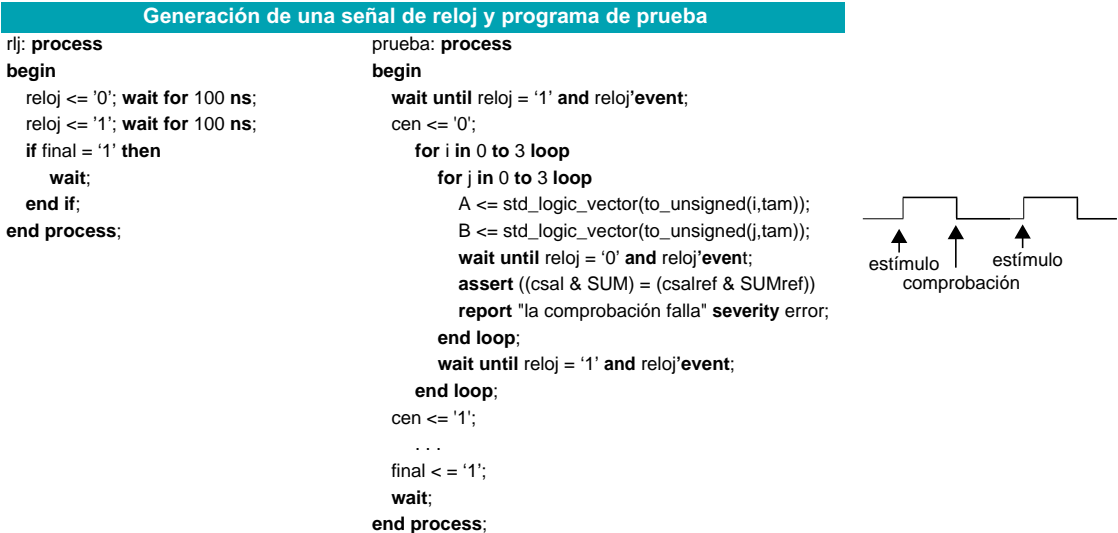


Figura 2.46 Utilización de una señal de reloj en un programa de prueba. El objeto final ha sido declarado como signal y ha sido inicializado a cero.

34. Este estilo de diseño del programa de prueba también es de utilidad en un diseño síncrono.
35. El atributo "event" es un atributo para los objetos de tipo señal. La función lógica en las sentencias "wait until" permiten detectar un flanco descendente y un flanco ascendente. En la siguiente sesión se detalla en mayor medida este atributo.

Trabajo: En el directorio SUMA/SUMGENERA/PRUEBAS se suministra un fichero que permite efectuar la comprobación de forma automática utilizando un modelo de referencia (prueba_snbits_reloj.vhd). Para ello, también se suministra un fichero con el modelo referencia (snref.vhd),

En el código del programa de prueba se utilizan funcionalidades de VHDL, que se corresponden con VHDL 2008, que no han sido explicadas, como "to_string", la cual convierte en una cadena de caracteres el valor de un objeto de un tipo, como por ejemplo "time" o "std_logic_vector". Para activar estas funcionalidades hay que indicarlo en Quartus para que se lo notifique a Modelsim. Para ello, cuando se indican los ficheros que deben utilizarse en la simulación (Figura 2.26, Figura 2.27), se selecciona el fichero que se quiere compilar en VHDL 2008. Posteriormente se pulsa en "Properties". Emerge la ventana que se muestra a la derecha de la Figura 2.47. En esta ventana hay que seleccionar VHDL 2008 en "HDL version".

Efectúe una simulación con los ficheros que se suministran. Formatee la ventana temporal para ayudar en el análisis de los resultados. Analice los resultados, tanto en la ventana temporal como en la ventana textual. El análisis de retardo en la ventana temporal se explica en el Apéndice 2.5.

En el fichero suministrado se calcula el retardo del sumador para cada conjunto de valores de entrada. Para ello se utiliza la función "now", que devuelve el tiempo de simulación actual, y el atributo "last_event", que devuelve el tiempo transcurrido desde el último evento observado en la señal a la que se aplica.

Modifique el fichero suministrado para calcular el valor mínimo y máximo del retardo del sumador, comprobando todos los posibles casos de las entradas. Imprima en la ventana textual (utilizando la sentencia "report") el primer valor de las entradas del sumador (teniendo en cuenta el orden de iteración de los bucles) en el que se observa el retardo mínimo y el máximo. Además del valor de las entradas imprima el retardo.

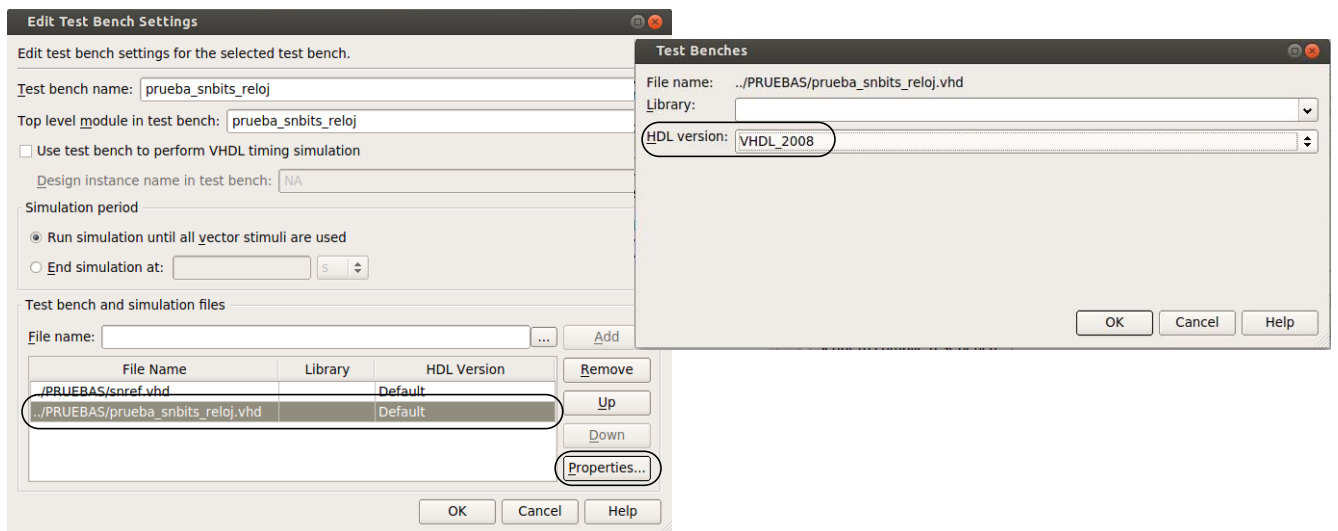


Figura 2.47 *Compilación del programa de prueba en VHDL 2008.*

Preguntas

- 3 Considere el sumador de 4 bits especificado en VHDL mediante sentencias generate (page 83). Tenga en cuenta que, en este diseño, los parámetros asociados a los retardos de las puertas se establecen al instanciar los componentes s1bits en el fichero snbits.vhd. Modifique el programa de pruebas (prueba_snbits_reloj.vhd)

Apéndice 2.1: Conversión y asimilación de tipo en el package numeric_std.all

Los tipos de objetos más comunes cuando se sintetiza una descripción VHDL (generación del hardware) son: std_logic, std_logic_vector, signed, unsigned e integer.

El lenguaje VHDL es un lenguaje fuertemente tipado. En consecuencia, los objetos que se utilizan en una expresión deben ser del mismo tipo.

En la Figura 2.48 se muestra la forma de convertir y asimilar objetos de tipos distintos. La asimilación de tipo se utiliza cuando los objetos fuente y destino se especifican con el mismo número de bits (std_logic_vector, signed, unsigned). Una función de conversión se utiliza cuando en el objeto fuente o destino no se ha especificado el número de bits. En concreto, el tipo integer no se declara con un número específico de bits³⁶. Notemos que en la conversión de integer a “signed” o “unsigned” se especifica el número de bits. Por otro lado, la conversión entre integer y std_logic_vector o viceversa requiere pasar por el tipo “signed” o el tipo “unsigned”. Esto es, no hay una conversión directa.

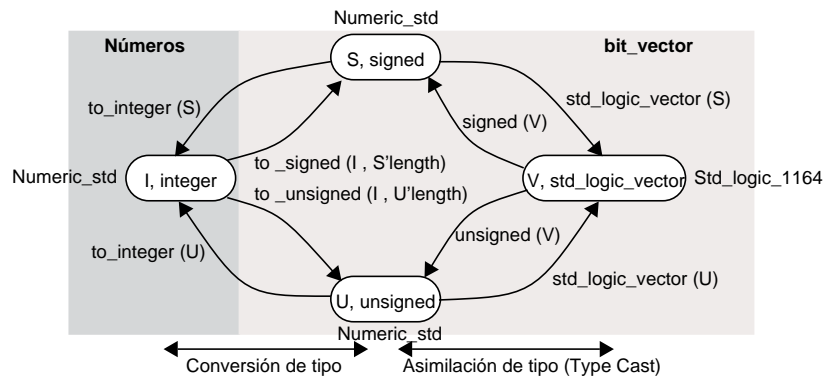


Figura 2.48 Conversión y asimilación de tipo.

Un ejemplo típico de conversión de un ítem std_logic_vector a integer es para indexar un elemento de un vector std_logic_vector. Un vector solo se puede indexar con enteros.

```
signal indice : std_logic_vector(2 downto 0);
signal A : std_logic_vector(7 downto 0);

... <= A(to_integer(unsigned(indice)));
```

Cuando en una operación aritmética los operandos son de distinto tamaño, se extiende la longitud del operando de menor longitud. Para la extensión se tiene en cuenta el tipo y signo del operando. En la Figura 2.49 se muestra un ejemplo.

36. Puede especificarse un rango de valores.

Tipo	Operación	Resultado
signed	"01101" + "1011"	"01101" + "11011" = "01000"
unsigned	"01101" + "1011"	"01101" + "01011" = "11000"

Figura 2.49 Suma de operandos con un número distinto de bits.

Cuando se efectúa la suma³⁷ de operandos "signed" o "unsigned", el acarreo de la suma de los bits más significativos se descarta. Si el acarreo es necesario, se debe añadir un bit extra a uno de los operandos (Figura 2.24).

	Operación	Resultado
constant A: unsigned (3 downto 0) := "1101";	Sumu <= '0' & A + unsigned ("0101");	"10010" (suma = 2, acarreo = 1)
constant B: signed (3 downto 0) := "1011";	Sums <= B(3) & B + signed ("1101");	"11000" (suma = -8, acarreo = 1)
signal Sumu: unsigned (4 downto 0);	Desbor <= Sums(4) /= Sums(3);	No hay desbordamiento. El resultado es representable.
signal Sums: signed (4 downto 0);		
signal Desbor: boolean		
signal X, Z: std_logic_vector (3 downto 0) := "0010";	Z <= std_logic_vector (A + unsigned (X));	"0000" (suma = 0, se descarta el acarreo)

Figura 2.50 Ejemplo de sumas.

Operador de relación. Un operador de relación compara dos operandos. La comparación se efectúa para determinar la igualdad o desigualdad y se tiene en cuenta el orden de especificación. La igualdad y desigualdad está predefinida para todos los tipos y el resultado es un valor booleano.

Operador	Semántica
/=	desigualdad
=	igualdad

37. Los operandos de las operaciones aritméticas definidas en este package deben ser objetos de tipo signed, unsigned o integer.

Apéndice 2.2: Instanciación de entidades y creación de librerías de recursos

Entidades en librerías de recursos. Cuando la entidad está incluida en una librería se utilizan las cláusulas “library” y “use” en la cabeza del fichero para indicar la librería.

Sintaxis	Sintaxis
library logical-library-name;	library logical-library-name;
use logical-library-name.all;	use logical-library-name.my_component;

A continuación se muestran dos posibilidades. En la declaración de la parte derecha solo es visible el componente que se quiere instanciar. Por ejemplo, la librería se denomina “Libs” y el componente se denomina S1bit.

library Libs;	library Libs;
use Libs.all;	use Libs.S1bit;

Creación de librería de recursos en Quartus. En Quartus se especifican todos los ficheros (sean o no de librería) en la página 2 al crear el proyecto (Figura 2.51)³⁸.

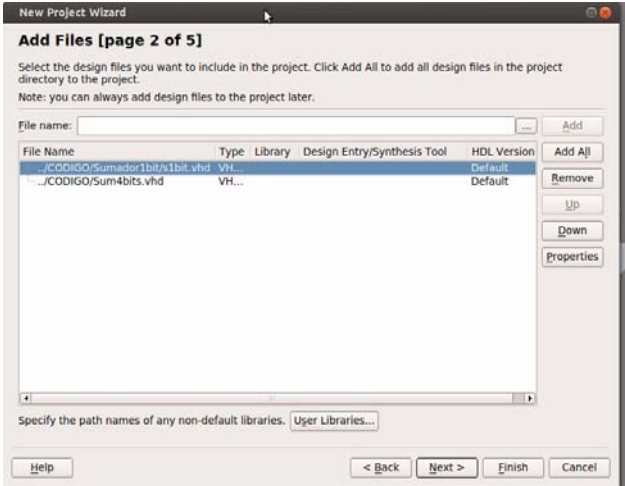


Figura 2.51 Especificación de los ficheros de un proyecto.

Una vez han sido especificados los ficheros, que se utilizan en el proyecto, hay que indicar que algunos de ellos estarán en una librería. Para ello en la pestaña “Files de “Project Navigator” hay que seleccionar el fichero que se quiere incluir en una librería y

38. Una alternativa, una vez creado el proyecto, es dar la orden “Assignments -> Files” y en la ventana emergente se añaden los ficheros.

pulsar el botón derecho del ratón. Entonces emerge una ventana en la que hay que seleccionar “Properties ...”. En la ventana que emerge posteriormente hay que indicar el nombre de la librería (Figura 2.52).

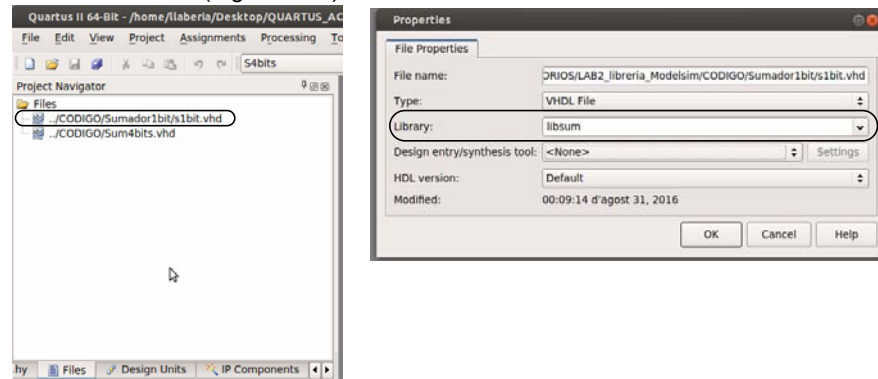


Figura 2.52 Inclusión de entidades en una librería.

Una vez se ha indicado en qué librería se quieren incluir los ficheros se da la orden “Processing -> Start -> Start Analysis & Elaboration” para compilar. Cuando finaliza el procesado de la orden, en la pestaña “Hierarchy” se observa la jerarquía del diseño y en la pestaña “Design Units” se observa la estructura en librerías (Figura 2.53).

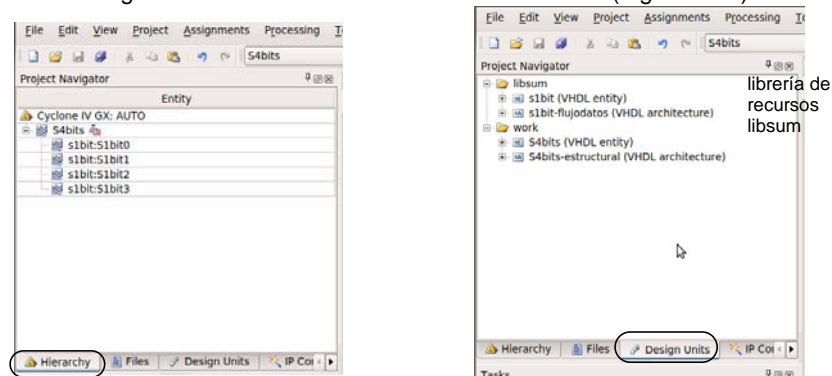


Figura 2.53 Pestañas “Hierarchy” y “Design Units” en un proyecto donde han sido especificadas librerías.

Simulador Modelsim. En la ventana “Library” de Modelsim se observan las librerías “work” y “libsum” (Figura 2.54).

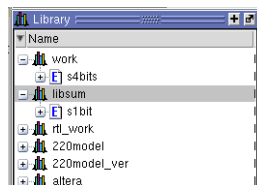


Figura 2.54 Ventana “Library” de Modelsim. Librerías “work” y “libsum”.

Instanciación directa de entidades

Instanciación de una entidad. Cada instancia de un módulo tiene un nombre que la identifica de forma unívoca y está seguida por la palabra clave entity y posteriormente por el nombre del componente. El nombre del componente precede a la palabra clave “port map”, a partir de la cual se especifica la asociación de señales entre niveles de la jerarquía. La sintaxis para la instanciación de un componente es la siguiente.

Sintaxis

```
instance_name: entity component_name
port map (port1 => signal1, port2 => signal2, . . . , portn => signaln);
```

En primer lugar se especifica el nombre de la instancia seguido de dos puntos, la palabra clave “entity” y el nombre del componente y la palabra clave “port map”. El nombre de la instancia o etiqueta puede ser cualquier identificador legal y es el nombre de la instancia concreta.

Seguidamente se muestra la codificación de un sumador de 4 bits donde la entidad s1bit está en la librería libsum.

<pre>library IEEE; use IEEE.std_logic_1164.all; library libsum; use libsum.all; -- La especificación del componente en VHDL se obtiene de una librería -- Los componentes se conectan mediante señales entity S4bits is port (A: in std_logic_vector (3 downto 0) ; B: in std_logic_vector (3 downto 0) ; cen: in std_logic; SUM: out std_logic_vector (3 downto 0) ; csal: out std_logic); end S4bits; architecture estructural of S4bits is signal c1, c2, c3, c4: std_logic; begin S1bit0: entity s1bit port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0)); S1bit1: entity s1bit port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1)); S1bit2: entity s1bit port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2)); S1bit3: entity s1bit port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3)); csal <= c4; end estructural;</pre>	<div>Librerías</div> <div>interfaz</div> <div>Listado de conexiones</div>
---	---

Sumador de 4 bits

Instanciación indirecta de entidades mediante componentes

Los componentes se declaran dentro del cuerpo de la arquitectura, siguiendo a la declaración de ésta. Esto es, después de la palabra clave “architecture” y antes de la palabra clave “begin”.

Declaración de los componentes. La forma de efectuar la declaración de los módulos que se van a utilizar es describir su interface. En este caso se utiliza la palabra clave “component”. Seguidamente se especifican las señales de entrada y salida, de la misma forma que se ha efectuado en la declaración de la interface (“entity”) al describir el módulo. La declaración finaliza con las palabras clave “end component”. En resumen, la declaración de un componente consiste del nombre del componente y de la interface (puertos), siendo la sintaxis la siguiente.

Sintaxis
<pre>component component_name [port (port_signal_name: mode type; port_signal_name: mode type; ... port_signal_name: mode type);] end component;</pre>

El nombre de un componente se refiere al nombre de una entidad definida explícitamente en un fichero VHDL. La lista de puertos de la interface especifica el nombre, el modo y tipo de cada puerto de forma idéntica a como se especifica en la declaración “entity”. En el fichero donde se declaran los componentes hay que indicar que el fichero que contiene la especificación del componente se almacena en el directorio de trabajo. Esta especificación se efectúa en la cabecera del fichero.

Sintaxis	Sintaxis
<pre>use work.all;</pre>	<pre>use work.my_component;</pre>

Nombre del componente que contiene el componente. El nombre del fichero con extensión “.vhd” que contiene la descripción del componente que se quiere utilizar tiene que ser el mismo que el nombre especificado en “entity”.

En la especificación mostrada a la izquierda se indica que son accesibles todas las definiciones existentes en el directorio de trabajo. En la especificación de la derecha se indica que solo son accesibles las especificaciones incluidas en el fichero denominado my_component.vhd.

Seguidamente se muestra la codificación de un sumador de 4 bits donde la entidad S1bit está en el fichero S1bit.vhd.

<pre>library IEEE; use IEEE.std_logic_1164.all; use work.all; -- La especificación del componente en VHDL está en otro fichero del directorio -- Los componentes se conectan mediante señales entity S4bits is generic (retardoxor: time := 15 ns; retardoand: time := 10 ns; retardoor: time := 15 ns); port (A: in std_logic_vector (3 downto 0) ; B: in std_logic_vector (3 downto 0) ; cen: in std_logic; SUM: out std_logic_vector (3 downto 0) ; csal: out std_logic); end S4bits; architecture estructural of S4bits is component S1bit generic(retardoxor: time, retardoand: time, retardoor: time) port (x, y, cen : in std_logic; s, csal : out std_logic); end component; signal c1, c2, c3, c4: std_logic; begin S1bit0: S1bit port map(x=>A(0), y=>B(0), cen=>cen, csal=>c1, s=>SUM(0)); S1bit1: S1bit port map(x=>A(1), y=>B(1), cen=>c1, csal=>c2, s=>SUM(1)); S1bit2: S1bit port map(x=>A(2), y=>B(2), cen=>c2, csal=>c3, s=>SUM(2)); S1bit3: S1bit port map(x=>A(3), y=>B(3), cen=>c3, csal=>c4, s=>SUM(3)); csal <= c4; end estructural;</pre>	Librerías
	interfaz
	Componentes
	Listado de conexiones
Seguidamente se muestra la declaración “entity” del componente S1bit.	
<pre>library IEEE; use IEEE.std_logic_1164.all; entity S1bit is generic (retardoxor: time := 15 ns; retardoand: time := 10 ns; retardoor: time := 10 ns); port (x: in std_logic; y: in std_logic ; cen: in std_logic; s: out std_logic; csal: out std_logic); end s1bit;</pre>	Librerías
	interfaz

Apéndice 2.3: Comprobación del funcionamiento lógico

Comprobar de forma exhaustiva el funcionamiento lógico de un circuito con un número significativo de bits en las entradas es una ardua tarea. El número de combinaciones posibles crece exponencialmente con el número de bits.

En su defecto utilizaremos un conjunto limitado de combinaciones de entrada para comprobar el funcionamiento lógico de un sumador de 4 bits³⁹.

Con el objetivo de que las señales tengan suficiente tiempo para propagarse de las entradas a las salidas, utilizaremos un intervalo de tiempo de 200 ns entre cambios en las señales de entrada.

En la Figura 2.55 se muestra una tabla donde se han especificado varios vectores de comprobación para el sumador de 4 bits. Los vectores de bit (A, B, SUM) están especificados en hexadecimal.

Entradas			Salidas	
A	B	cen	SUM	csal
0	0	0	0	0
0	4	0	4	0
0	4	1	5	0
0	a	1	b	0
2	a	1	d	0
5	2	1	8	0
1	1	1	3	0

Figura 2.55 Muestra de vectores de comprobación para el sumador de 4 bits.

Partiendo de la tabla de la Figura 2.55, en el programa de prueba se especifican las sentencias de asignación de señal para generar los frentes de onda, que se utilizan como estímulos, como se muestra en la Figura 2.56. Para especificar un valor en hexadecimal se utiliza el símbolo X como prefijo, seguido de " (doble comilla), el valor numérico en hexadecimal y ".

```
a <= x"0" after 100 ns, x"0" after 300 ns, x"0" after 500 ns, x"0" after 700 ns, x"2" after 900 ns, x"5" after 1100 ns, x"1" after 1300 ns;
b <= x"0" after 100 ns, x"4" after 300 ns, x"4" after 500 ns, x"A" after 700 ns, x"A" after 900 ns, x"2" after 1100 ns, x"1" after 1300 ns;
cen <= '0' after 100 ns, '0' after 300 ns, '1' after 500 ns, '1' after 700 ns, '1' after 900 ns, '1' after 1100 ns, '1' after 1300 ns;
```

Figura 2.56 Estímulos de los puertos de entrada.

En la Figura 2.57 se muestra la traza de las señales en la ventana temporal después de finalizar la simulación. Las entradas de interés se identifican con óvalos y las salidas correspondientes con rectángulos. Las salidas se corresponden con los valores esperados (Figura 2.55).

39. Este conjunto de combinaciones de entrada debe diseñarse con criterio. Se parte de que el sumador de un bit ha sido comprobado de forma exhaustiva. En el sumador de 4 bits hay que comprobar la propagación y absorción del acarreo.

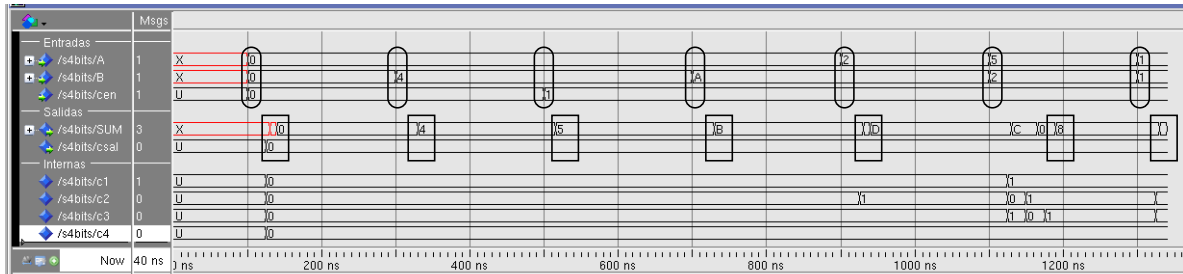


Figura 2.57 Ventana temporal. Simulación lógica.

En la tabla de la Figura 2.58 se muestra la salida textual. En esta tabla se indica con sombreado el instante de tiempo en el cual se modifica una entrada. En la 5ª y 6ª columna se observa el resultado recuadrado.

ps	/s4bits/ A	/s4bits/ B	/s4bits/ cen	/s4bits/ SUM	/s4bits/ csa1	/s4bits/ c1	/s4bits/ c2	/s4bits/ c3	/s4bits/ c4
0	UUUU	UUUU	U	UUUU	U	U	U	U	U
100000	0000	0000	0	UUUU	U	U	U	U	U
125000	0000	0000	0	UUUU	0	0	0	0	0
130000	0000	0000	0	UUUU	0	0	0	0	0
140000	0000	0000	0	0000	0	0	0	0	0
300000	0000	0100	0	0000	0	0	0	0	0
330000	0000	0100	0	0100	0	0	0	0	0
500000	0000	0100	1	0100	0	0	0	0	0
515000	0000	0100	1	0101	0	0	0	0	0
700000	0000	1010	1	0101	0	0	0	0	0
730000	0000	1010	1	1011	0	0	0	0	0
900000	0010	1010	1	1011	0	0	0	0	0
925000	0010	1010	1	1011	0	0	1	0	0
930000	0010	1010	1	1001	0	0	1	0	0
940000	0010	1010	1	1101	0	0	1	0	0
1100000	0101	0010	1	1101	0	0	1	0	0
1125000	0101	0010	1	1101	0	1	0	1	0
1130000	0101	0010	1	1100	0	1	0	1	0
1150000	0101	0010	1	1100	0	1	1	0	0
1165000	0101	0010	1	0000	0	1	1	0	0
1175000	0101	0010	1	0000	0	1	1	1	0
1190000	0101	0010	1	1000	0	1	1	1	0
1300000	0001	0001	1	1000	0	1	1	1	0
1325000	0001	0001	1	1000	0	1	0	0	0
1330000	0001	0001	1	1011	0	1	0	0	0
1340000	0001	0001	1	0011	0	1	0	0	0

Figura 2.58 Salida textual. Simulación lógica. La flecha indica relación entre entradas y salidas.

Apéndice 2.4: Ejemplo. Utilización de la sentencia “generate”

En la Figura 2.59 se muestra una especificación de un sumador de 4 bits mediante la sentencia “generate” utilizando como componente un sumador de 1 bit.

Sumador de 4 bits	Sumador de 1 bit (componente)
<pre> library ieee; use ieee.std_logic_1164.all; use work.all; entity snbits is generic (n: positive:= 4); port (a: in std_logic_vector (n-1 downto 0); b: in std_logic_vector (n-1 downto 0); cen: in std_logic; s: out std_logic_vector (n-1 downto 0); csal: out std_logic); end snbits; architecture Gestructural of snbits is component s1bit generic(retardoxor: time ; retardoand: time ; retardoor: time); port (x: in std_logic; y: in std_logic; cen: in std_logic; s: out std_logic; csal: out std_logic); end component; signal c : std_logic_vector (n downto 0); begin c(0) <= cen; sumador: for i in 0 to n-1 generate sumi: s1bit generic map(retardoxor => 15 ns, retardoand => 10 ns, retardoor => 15 ns) port map (x => a(i), y => b(i), cen => c(i), s => s(i), csal => c(i+1)); end generate; csal <= c(n); end Gestructural; </pre>	<pre> library ieee; use ieee.std_logic_1164.all; entity s1bit is generic(retardoxor: time := 5 ns; retardoand: time := 2 ns; retardoor: time := 5 ns); port (x: in std_logic; y: in std_logic; cen : in std_logic; s: out std_logic; csal: out std_logic); end s1bit; architecture flujodatos of s1bit is signal xorxy, andxy, andxcen, andycen : std_logic; begin xorxy <= x xor y after retardoxor; s <= xorxy xor cen after retardoand; andxy <= x and y after retardoxor; andxcen <= x and cen after retardoand; andycen <= y and cen after retardoand; csal <= andxy or andxcen or andycen after retardoor; end flujodatos; </pre>

Figura 2.59 Versión 1. Generación de un sumador de 4 bits.

En la Figura 2.60 se muestran dos versiones más para generar un sumador de 4 bits. En las dos se utiliza como esquema de generación “if” además de “for”. La diferencia entre ellas reside en la forma de especificar el rango de valores en la generación. En la parte izquierda se utiliza directamente el parámetro genérico que especifica el tamaño. En la parte derecha se utilizan atributos predefinidos en VHDL para las señales.

Atributos. Los atributos suministran información adicional de un ítem. Por ejemplo, de un signal o de objetos tipo array. La sintaxis de un atributo es el nombre del ítem seguido de un apóstrofe y la identificación del atributo.

Sintaxis

object'attribute_name

Sumador de 4 bits

Ejemplos de atributos.

atributos (A es un array)		Ejemplos
A'left	es el índice más a la izquierda que identifica un elemento del array	signal A: std_logic_vector (7 downto 0);
A'right	es el índice más a la derecha que identifica un elemento del array	signal B: std_logic_vector (0 to 7);
A'range	es el rango A'left to A'right o A'left downto A'right	A'left: A(7)
A'length	es el número de elementos de array. A'left - A'right +1	B'left: B(0)
		A'right: A(0)
		B'right: B(7)
		A'range: 7 downto 0
		B'range: 0 to 7
		A'length: 8
		B'length: 8

En la Figura 2.61 se muestra el diseño RTL que genera Quartus a partir de la especificación de la derecha de la Figura 2.60.

Sumador de 4 bits	Utilización de atributos
<pre>library ieee; use ieee.std_logic_1164.all; use work.all; entity snbits is generic (n: positive:= 4); port (a: in std_logic_vector (n-1 downto 0); b: in std_logic_vector (n-1 downto 0); cen: in std_logic; s: out std_logic_vector (n-1 downto 0); csal: out std_logic); end snbits; architecture Gestructural of snbits is component s1bit generic(retardoxor: time ; retardoand: time ; retardoor: time); port (x: in std_logic; y: in std_logic; cen: in std_logic; s: out std_logic; csal: out std_logic); end component; signal c : std_logic_vector (n downto 0); begin sumador: for i in 0 to n-1 generate cero: if (i = 0) generate sum0: s1bit generic map(retardoxor => 15 ns, retardoand => 10 ns, retardoor => 15 ns) port map (x => a(i), y => b(i), cen => cen, s => s(i), csal => c(i+1)); end generate; sumin: if (i > 0 and i < n-1) generate sumi: s1bit generic map(retardoxor => 15 ns, retardoand => 10 ns, retardoor => 15 ns) port map (x => a(i), y => b(i), cen => c(i), s => s(i), csal => c(i+1)); end generate; ultimo: if i = n-1 generate sumul: s1bit generic map(retardoxor => 15 ns retardoand => 10 ns, retardoor => 15 ns) port map (x => a(i), y => b(i), cen => c(i), s => s(i), csal => csal); end generate; end generate; end Gestructural;</pre>	<div>sumador: for i in a'range generate cero: if (i = a'right) generate</div> <div>sumin: if (i > a'right and i < a'left) generate</div> <div>ultimo: if i = a'left generate</div>

Figura 2.60 Versiones 2 y 3. Generación de un sumador de 4 bits.

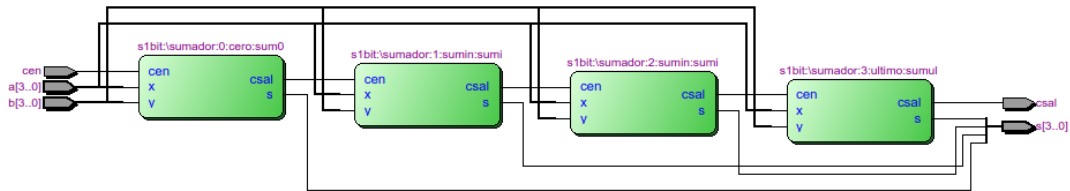


Figura 2.61 Esquema RTL de la especificación derecha de la Figura 2.60.

Apéndice 2.5: Medida del tiempo de retardo

En este apéndice se muestran dos formas de medir el retardo de forma manual. En primer lugar se muestra cómo medir el retardo para un conjunto de vectores de entrada. Posteriormente se muestra cómo utilizar frentes de onda para estimular los puertos de entrada y medir retardos.

Simulación temporal del circuito

Cuando se activa la simulación en Modelsim las señales en los puertos de entrada están indefinidas. Entonces, si establecemos valores en los puertos de entrada podemos evaluar el retardo para estas entradas.

Seguidamente se muestra, para el sumador de 4 bits, cómo medir el retardo y el análisis del mismo para un caso concreto: $A = 9$, $B = 8$ y $cen = 0$, habiéndose especificado A y B en hexadecimal.

Desde Quartus se activa la simulación del sumador de 4 bits sin la indicación de que utilice un programa de prueba. Posteriormente se dan las siguientes órdenes en la ventana de mensajes de Modelsim.

Órdenes

```
force -freeze sim:/estimulos_s4bits/A 16#9 0
force -freeze sim:/estimulos_s4bits/B 16#8 0
force -freeze sim:/estimulos_s4bits/cen 0 0
run -all
```

Después de añadir cursores y acomodar la visión de la ventana temporal se observa la traza que se muestra en la Figura 2.62. En la Figura 2.63 se muestra la ventana textual. La codificación de las filas de la Figura 2.63 se ha explicado en la práctica 1.

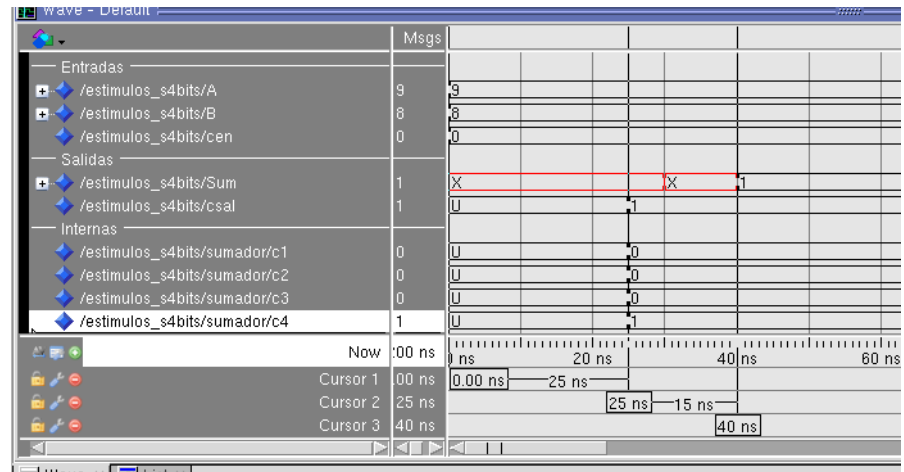


Figura 2.62 Medida de retardos. Traza en la ventana temporal. Sumador de 4 bits con valores de entrada: A=9, B=8 y cen=0.

Los cursores 2 y 3 están posicionados, respectivamente, en el instante de tiempo a partir del cual las señales `csal` y `Sum` son estables. La señal `csal` es estable al cabo de 25 ns y la señal `Sum` es estable al cabo de 40 ns. En consecuencia el retardo del sumador para estos valores de entrada es 40 ns. El análisis de la ventana textual corrobora los retardos deducidos de la ventana temporal (Figura 2.63).

ps	delta	/estimulos_s4bits/A	/estimulos_s4bits/sumador/c1	/estimulos_s4bits/B	/estimulos_s4bits/sumador/c2	/estimulos_s4bits/Sum	/estimulos_s4bits/sumador/c3	/estimulos_s4bits/cen	/estimulos_s4bits/sumador/c4	/estimulos_s4bits/csai
0	+0	1001 1000	UUUU 0 U	UUUU 0 U	UUUU 0 U	0	0	0	0	0
25000	+0	1001 1000	UUUU 0 U	UUUU 0 U	UUUU 0 U	0	0	0	0	0
25000	+1	1001 1000	UUUU 0 1	UUUU 0 1	UUUU 0 1	0	0	0	0	0
30000	+0	1001 1000	UUUU 0 1	UUUU 0 1	UUUU 0 1	0	0	0	0	0
40000	+0	1001 1000	0001 0 1	0001 0 1	0001 0 1	0	0	0	0	0

Figura 2.63 Medida de retardos. Ventana textual. Sumador de 4 bits con valores de entrada: A=9, B=8 y cen=0.

Para observar, en la ventana de tiempo, las señales individualizadas que transporta un bus hay que desagregar las señales. En la Figura 2.64 se muestra la ventana temporal después de pulsar en el símbolo “+” asociado a la señal `Sum`. Posteriormente se indica representación “literal” de las señales desagregadas. En esta figura se ha añadido un nuevo cursor para indicar el instante de tiempo a partir del cual `Sum(0)` es estable (cursor 4).

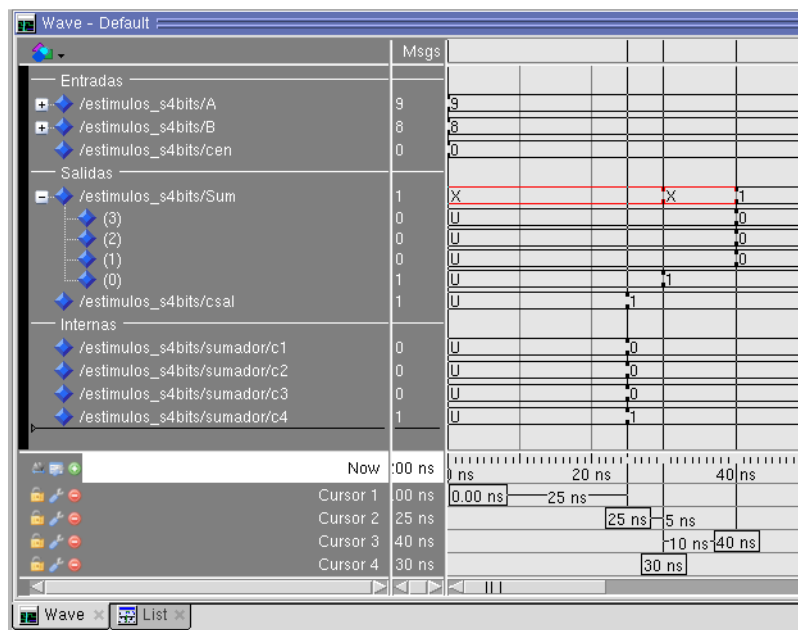


Figura 2.64 Medida de retardos. Traza en la ventana temporal. Sumador de 4 bits con valores de entrada: $A=9$, $B=8$ y $cen=0$. Observación de señales individuales del vector Sum.

Observemos que en $t = 30$ ns la señal Sum(0) es estable mientras que el resto de señales de la señal agregada Sum no están estabilizadas. Este comportamiento también se observa en la ventana textual (Figura 2.63). Notemos también que la señal Sum(0) se estabiliza después de 5 ns de que se haya estabilizado la señal csal. Para determinar el valor de la señal csal solo son necesarios, en este caso, las señales A(3) y B(3), ya que tienen el valor “1” y determinan directamente un acarreo de salida.

Frente de onda para la estimulación

La medida del retardo para distintos conjuntos de valores se puede automatizar utilizando frentes de onda en el programa de prueba. Cada frente de onda es similar al que se utiliza para comprobar el funcionamiento lógico. Entre vectores de estímulos cuyo retardo se quiere medir, se inserta un vector de estímulos con las entradas indefinidas.

En la tabla de la Figura 2.65 se muestra un ejemplo. Esta tabla se ha construido partiendo de la tabla utilizada para la comprobación del funcionamiento lógico del sumador de 4 bits (Figura 2.55).

Entradas			Salidas	
A	B	cen	SUM	csal
X	X	X	X	X
0	0	0	0	0
X	X	X	X	X
0	4	0	4	0
X	X	X	X	X
0	4	1	5	0
X	X	X	X	X
0	a	1	b	0
X	X	X	X	X
2	a	1	d	0
X	X	X	X	X
5	2	1	8	0
X	X	X	X	X
1	1	1	3	0

Figura 2.65 Tabla de estimulación para medir retardos.

Partiendo de la tabla de la Figura 2.65, en el programa de prueba se escriben las sentencias de asignación de señal que generan los frentes de onda, que se utilizan como estímulos, como se muestra en la Figura 2.66. Con el objetivo de que las señales tengan suficiente tiempo para propagarse de las entradas a las salidas, utilizaremos un intervalo de tiempo de 100 ns entre cambios en las señales de entrada.

```
a <= (others =>'U'), x"0" after 100 ns, (others =>'U') after 200 ns, x"0" after 300 ns, (others =>'U') after 400 ns, x"0" after 500 ns,
(others =>'U') after 600 ns, x"0" after 700 ns, (others =>'U') after 800 ns, x"2" after 900 ns, (others =>'U') after 1000 ns, x"5" after
1100 ns, (others =>'U') after 1200 ns, x"1" after 1300 ns
b <= (others =>'U'), x"0" after 100 ns, (others =>'U') after 200 ns, x"4" after 300 ns, (others =>'U') after 400 ns, x"4" after 500 ns,
(others =>'U') after 600 ns, x"A" after 700 ns, (others =>'U') after 800 ns, x"A" after 900 ns, (others =>'U') after 1000 ns, x"2" after
1100 ns, (others =>'U') after 1200 ns, x"1" after 1300 ns
cen <= 'U', '0' after 100 ns, 'U' after 200 ns, '0' after 300 ns, 'U' after 400 ns, '1' after 500 ns, 'U' after 600 ns, '1' after 700 ns, 'U' after
800 ns, '1' after 900 ns, 'U' after 1000 ns, '1' after 1100 ns, 'U' after 1200 ns, '1' after 1300 ns
```

Figura 2.66 Estímulos de los puertos de entrada para evaluar retardos.

En la Figura 2.67 se muestra la traza de las señales en la ventana temporal después de finalizar la simulación. Mediante la utilización de cursores se identifica la estimulación de los puertos de entrada con señales distintas del valor indefinido. Además se identifica el instante en que las salidas del sumador son estables.

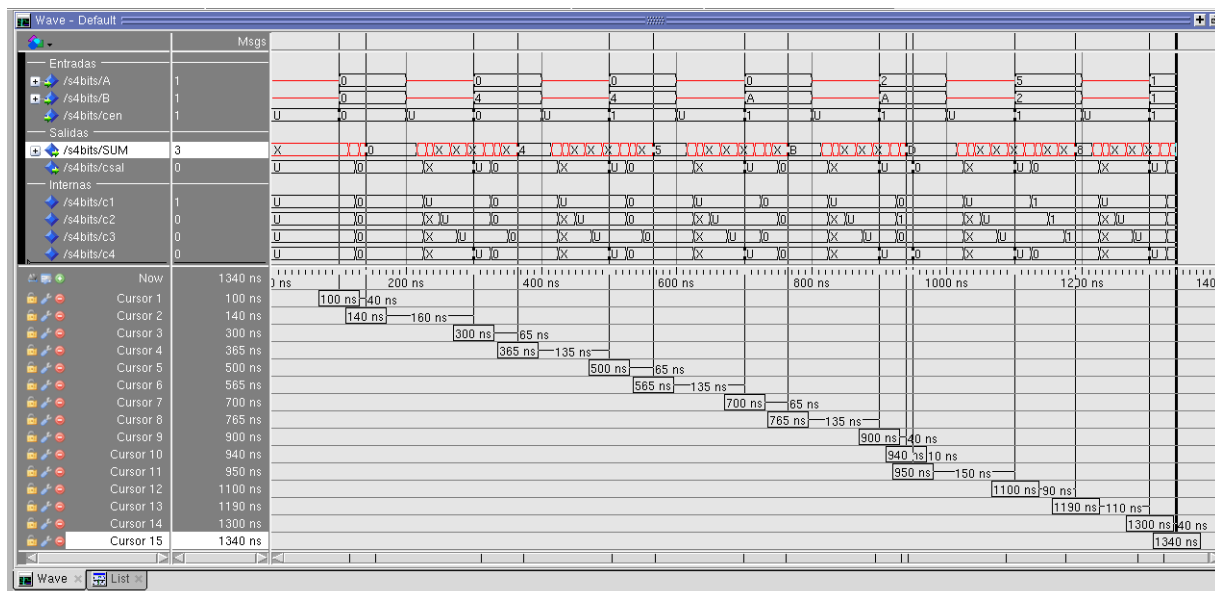


Figura 2.67 Trazas en la ventana temporal utilizando frentes de onda para evaluar retardos.

En la Figura 2.67 se observa que en $t = 100$ ns las entradas se estimulan con el vector (cen = 0, A = 0 y B = 0), interpretándose los últimos dos dígitos en hexadecimal (cursor 1 en la Figura 2.67). En $t = 140$ ns las salidas son estables (csal = 0 y Sum = 0, cursor 2 en la Figura 2.67). Entonces, el retardo es $140 - 100 = 40$ ns. Otro ejemplo es el vector de estímulos (0, 0, 4) el cual se aplica en $t = 300$ ns (cursor 3 en la Figura 2.67). Las salidas son estables en $t = 365$ ns (cursor 4 en la Figura 2.67). Entonces, el retardo son 65 ns. Finalmente comentamos el vector de estímulos (1, 2, A) que se aplica en $t = 900$ ns (cursor 9 en la Figura 2.67). Las salidas son estables en $t = 950$ ns (cursor 11 en la Figura 2.67). Por tanto, el retardo son 50 ns.

En la Figura 2.68 se muestra un conjunto de sentencias de asignación de señal para comprobar el sumador de 4 bits. Debido a que el retardo que se especifica en dichas sentencias, entre algunos cambios de valor en las señales, es menor que el retardo de propagación del circuito, el funcionamiento es incorrecto. El resultado de la simulación con estos estímulos se muestra en la Figura 2.69. Notemos que la salida no es correcta.

```
a <= "xxx", "0000" after 200 ns, "xxx" after 230 ns, "0000" after 430 ns, "xxx" after 460 ns, "0000" after 660 ns,
"xxx" after 690 ns;
b <= "xxx", "0000" after 200 ns, "xxx" after 230 ns, "0100" after 430 ns, "xxx" after 460 ns, "0100" after 660 ns,
"xxx" after 690 ns;
cen <= 'x', '0' after 200 ns, 'x' after 230 ns, '0' after 430 ns, 'x' after 460 ns, '1' after 660 ns, 'x' after 690 ns;
```

Figura 2.68 Sentencias de asignación de señal que estimulan el circuito a una velocidad mayor que la permitida.

En la Figura 2.70 se muestra la ventana textual y una ampliación del rango [200, 280] ns.

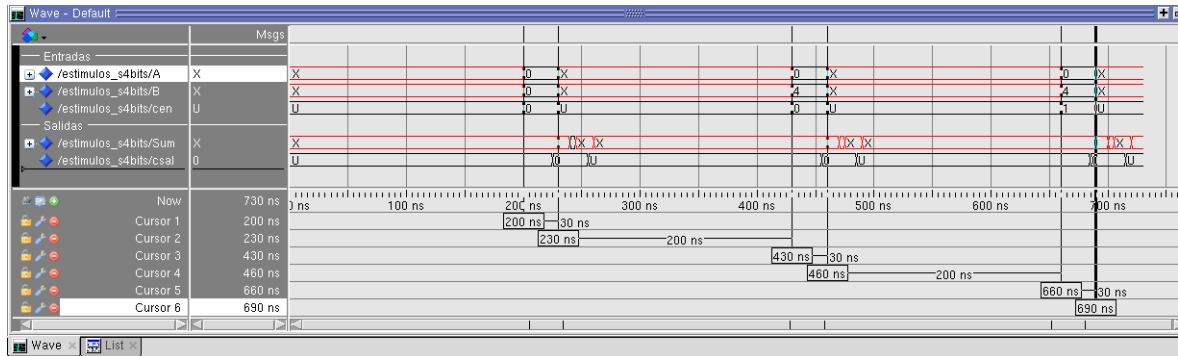


Figura 2.69 Trazo de la ventana de tiempos cuando la señales de entrada cambian más rápido que el retardo del circuito.

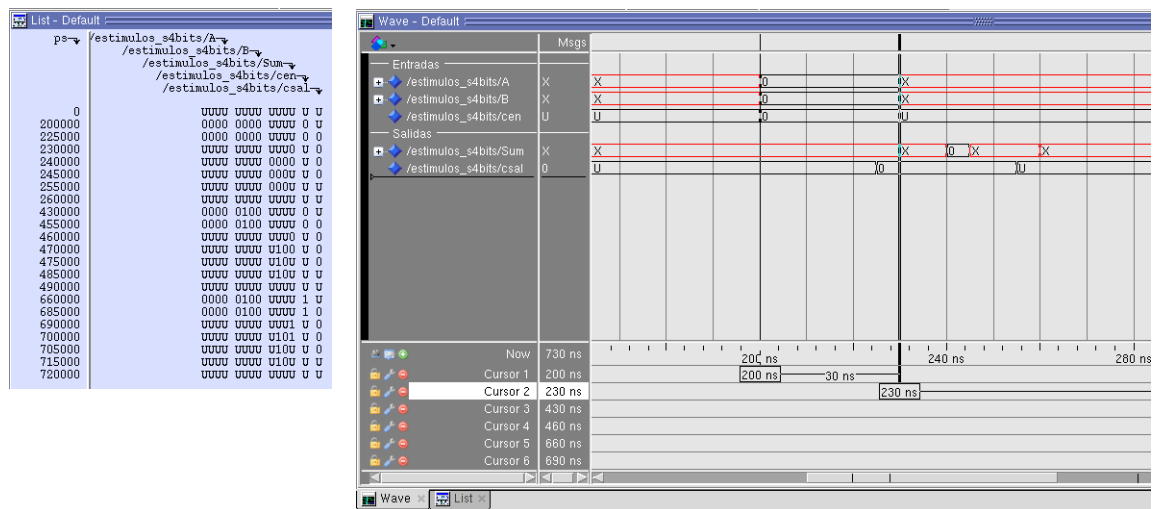


Figura 2.70 Ventana textual cuando las señales de entrada cambian más rápido que el retardo del circuito. Detalle de la ventana temporal en el rango [200, 280] ns.

Apéndice 2.6: Package

Un “package” es una funcionalidad de VHDL que permite compartir declaraciones entre varios diseños. Un “package” está incluido en una librería y en el diseño donde se utilice hay que hacerlo visible mediante la palabra clave “use”. Por ejemplo, si el “package”, denominado tipos, está en la librería “work”, en la parte declarativa de un diseño se especifica lo siguiente.

Declaraciones

```
library work;
use work.tipos.all;
```

Por otro lado, esta funcionalidad permite que el código sea más inteligible y fácil de modificar. Por ejemplo, no es necesario repetir las declaraciones en los distintos diseños donde se utilizan.

Por ahora un “package” se utilizará para ubicar declaraciones de constantes, tipos, funciones, procedimientos y componentes⁴⁰.

En un “package” distinguimos dos partes: la parte declarativa y el cuerpo. En la parte declarativa se especifican, por ejemplo, constantes, subtipos, funciones y procedimientos con los parámetros que se utilizan (prototipos) y componentes. En el cuerpo del “package” se especifica la implementación de las funciones y procedimientos declarados en la parte declarativa.

Sintaxis

```
package package_name is
    (declarations)
end package package_name;
package body package_name is
    (functions and procedure descriptions)
end package body package_name;
```

40. Esta funcionalidad es de utilidad cuando se utiliza un número significativo de componentes en un diseño estructural.

Apéndice 2.7: Multiplexor y decodificador

En este apéndice se describen los circuitos combinacionales multiplexor y decodificador, los cuales circuitos combinacionales básicos⁴¹. Además se describen otros constructores de VHDL que permiten describir modelos de comportamiento mediante sentencias de asignación condicional.

Multiplexor

Un multiplexor es un circuito que permite seleccionar una señal de salida de entre varias señales de entrada en función de una señal de entrada de control (señal de selección). La tabla de verdad de un multiplexor 4-1, selección de una salida de entre 4 entradas, se muestra en la parte izquierda de la Figura 2.71⁴². Las señales de entrada se denominan D0, D1, D2, D3. Las señales de control son S1 y S0. En un caso general, el número de señales de control de 1 bit es n, siendo 2^n el número de señales de entradas entre las que se puede efectuar una selección. En la parte central de la Figura 2.71 se muestran unas expresiones lógicas que se obtienen de la tabla de verdad. En la parte derecha se muestra el símbolo utilizado para representar un multiplexor.

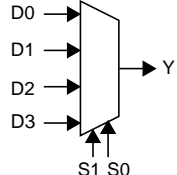
S1	S0	Y	Expresiones lógicas	Símbolo
0	0	D0	$Y = D0 \overline{S1} \overline{S0} + D1 \overline{S1} S0 + D2 S1 \overline{S0} + D3 S1 S0$	
0	1	D1		
1	0	D2		
1	1	D3		

Figura 2.71 Multiplexor: tabla de verdad, expresiones lógicas y símbolo en un esquema de circuito.

La señales de entrada de un multiplexor pueden ser grupos de bits (bus), en cuyo caso la selección es de todas las señales que transportan un bus.

En la Figura 2.72 se muestra un esquema con puertas lógicas de un multiplexor 4-1.

41. Algunos de ellos se utilizan en la siguiente sesión.

42. Notemos que es una tabla de verdad distinta de la convencional. En la columna de salida se especifica la entrada que se selecciona. Una tabla convencional tendría 6 variables (2 sel, 4 entradas) y para una combinación de las variables de selección tendríamos que, para las 2^4 combinaciones de las variables de entrada, sólo habría 2 salidas distintas que se corresponderían con los valores de una de las variable de entrada.

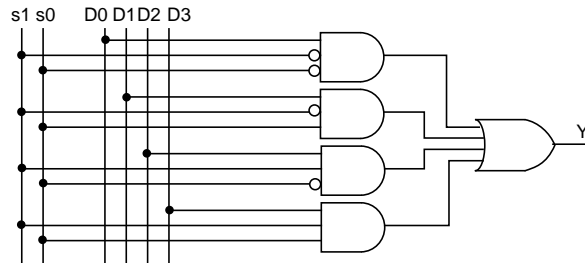


Figura 2.72 Esquema de puertas de un multiplexor. Los círculos en la entrada de una puerta indican señal negada.

Un multiplexor se puede utilizar para implementar funciones lógicas de forma sencilla. Una función de n variables de entrada tiene 2^n “minterm”⁴³ y cada “minterm” se corresponde con una de las 2^n entradas del multiplexor. Las n variables de entrada de la función lógica son las n líneas de selección del multiplexor. Una entrada del multiplexor es uno o cero dependiendo de si el correspondiente “minterm” de la función es uno o cero respectivamente.

Decodificador

Un decodificador es un bloque combinacional que convierte una forma de representación digital en otra. Usualmente, un decodificador tiene como entrada una representación compacta (codificada) y la convierte en una representación expandida (explícita).

Un decodificador binario de n entradas es un circuito combinacional que tiene n entradas y 2^n salidas. Dada una combinación de los valores de entrada sólo una de las salidas es 1 y las otras salidas son cero.

Un decodificador binario se utiliza siempre que un conjunto de valores ha sido codificado en binario y los valores tienen que distinguirse individualmente (decodificar). En general, un conjunto de 2^n elementos se puede codificar en n bits. Entonces, un decodificador de n bits se utiliza para identificar cuál de los elementos del conjunto ha sido codificado. Un ejemplo típico de utilización de un decodificador es en la decodificación de direcciones para acceder a posiciones de almacenamiento (memoria, banco de registros).

43. Recordemos que un “minterm” es un producto que contiene todas las variables utilizadas en la función lógica.

La tabla de verdad de un decodificador de 2 entradas se muestra en la parte izquierda de la Figura 2.73. Las señales de entrada se denominan S0, S1. Las señales de salida son D3, D2, D1 y D0. En la parte central de la Figura 2.73 se muestran unas expresiones lógicas que se obtienen de la tabla de verdad. En la parte derecha se muestra el símbolo utilizado para representar un decodificador.

S1	S0	D3, D2, D1, D0	Expresiones lógicas	Símbolo
0	0	0, 0, 0, 1	$D0 = \overline{S1} \overline{S0}$	
0	1	0, 0, 1, 0	$D1 = \overline{S1} S0$	
1	0	0, 1, 0, 0	$D2 = S1 \overline{S0}$	
1	1	1, 0, 0, 0	$D3 = S1 S0$	

Figura 2.73 Decodificador: expresiones lógicas y símbolo en un esquema de circuito.

En la Figura 2.74 se muestra el esquema con puertas lógicas de un decodificador de 2 entradas.

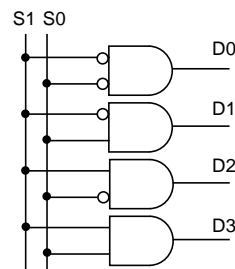


Figura 2.74 Esquema de puertas de un decodificador de dos entradas. Los círculos en la entrada de una puerta indican señal negada.

Un decodificador binario y una puerta OR permiten implementar cualquier expresión lógica. Esto es una consecuencia de la definición de un decodificador binario. Una salida del decodificador representa una de las posibles combinaciones de las señales de entrada ("minterm"), entonces, la implementación de suma de "minterms" se obtiene efectuando la OR de las salidas del decodificador que se corresponde con los "minterm" de la expresión lógica.

Decodificador - multiplexor

Un multiplexor es una extensión de un decodificador en el que una serie de entradas son decodificadas para obtener las señales de selección que permiten elegir una señal de un conjunto de señales de entrada.

De forma similar a como en un decodificador se utilizan n bits para decodificar 2^n señales, en un multiplexor con 2^n señales de entrada son necesarios n bits para líneas de selección.

En la Figura 2.75 se muestra un multiplexor donde se distingue un primer nivel de puertas que conforma un decodificador. La salida es un vector de bits donde sólo uno de los bits toma el valor uno. Los dos siguientes niveles de puertas encaminan a la salida la señal seleccionada en el decodificador.

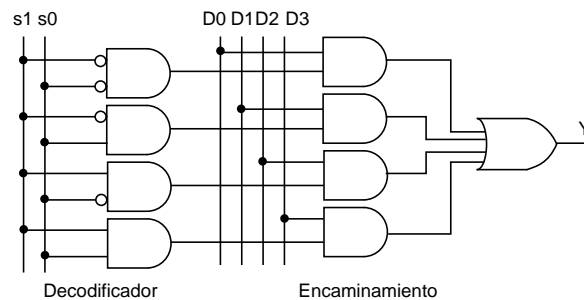


Figura 2.75 Esquema de puertas lógicas de un multiplexor diferenciando una primera parte de decodificación y una segunda parte de encaminamiento.

Puerta de tres estados

En una puerta de tres estados, además de los 2 valores usuales 1 y 0, la salida puede tomar el valor Z (alta impedancia, no conectado). La puerta tiene dos entradas: dato de entrada y permiso ("enable"). La salida es igual a la entrada cuando la señal permiso es 1, mientras que la salida no está conectada si la señal permiso es cero (Figura 2.76).

permiso	sal	Símbolo
0	Z (alta impedancia)	
1	ent	

Figura 2.76 Puerta de tres estados: tabla de verdad y símbolo en un esquema de circuito.

Las salidas de varias puertas de tres estados se pueden conectar juntas mientras sólo una de ellas tenga permiso en un momento dado. Por tanto se utilizan para conectar varios dispositivos al mismo bus.

Multiplexor. En la Figura 2.77 se muestra la utilización de puertas de tres estados para construir un multiplexor. El primer nivel de puertas es un decodificador. La salida de este decodificador determina cuál de las puertas de tres estados no está en alta impedancia.

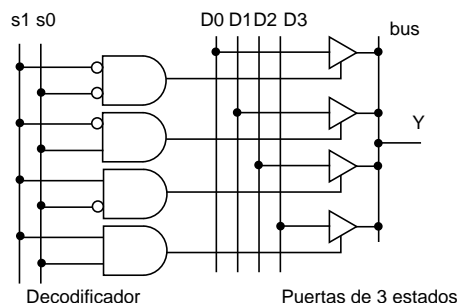


Figura 2.77 Multiplexor utilizando un decodificador y puertas de tres estados.

Constructores condicionales de asignación de señal en VHDL

Un circuito combinacional se puede describir a nivel de puertas y tendremos un modelo estructural. También se puede describir el comportamiento. Para ello en VHDL podemos utilizar sentencias de asignación de señal concurrentes. El modelo usualmente se denomina de flujo de datos. Recordemos que un modelo de flujo de datos describe un circuito en términos de su función y el flujo de datos a través del circuito.

En VHDL se pueden utilizar varios constructores para describir modelos de comportamiento. En esta sección nos centraremos en sentencias de asignación de señal concurrentes tanto simples como con capacidad para determinar una de varias posibilidades de asignación: a) sentencias de asignación de señal concurrentes, b) selección en la asignación de una señal (selected signal assignments) y c) asignación de una señal de forma condicional (conditional signal assignments).

Las sentencias de asignación de señal concurrentes se activan y ejecutan tan pronto se produce un evento en una de las señales. Esto es, una señal cambia de valor.

Sentencia de asignación de señal concurrente simple

Anteriormente ya se han visto varios ejemplos donde se utilizaban sentencias de asignación de señal concurrentes. En este apartado repasaremos la sentencia de asignación de señal básica. La sintaxis es la siguiente.

Ejemplo	Sintaxis
<code>s <= (x xor y) xor cen;</code>	<code>signal-name <= expresion;</code>

El valor evaluado en la expresión se transfiere a la señal. Tan pronto como se produce un evento (cambio en el valor) en una de las señales utilizadas en la expresión, la expresión se evalúa. El tipo de la señal a la que se asigna el valor debe ser el mismo que el tipo del valor que se evalúa en la expresión. Utilizando las expresiones lógicas que se obtienen de la tabla de verdad del multiplexor y del decodificador podemos escribir los códigos en VHDL que se muestran en la Figura 2.78.

Multiplexor	Decodificador
<pre> library IEEE; use IEEE.std_logic_1164.all; entity mux4 is port (d0, d1, d2, d3: in std_logic; s0, s1: in std_logic; y: out std_logic); end mux4; architecture behavB of mux4 is begin y <= (d0 and (not s1) and (not s0)) or (d1 and (not s1) and s0) or (d2 and s1 and (not s0)) or (d3 and s1 and s0); end behavB; </pre>	<pre> library IEEE; use IEEE.std_logic_1164.all; entity decodificador is port (sel0, sel1 : in std_logic; d0, d1, d2, d3: out std_logic); end decodificador; architecture behavS of decodificador is signal sel0_n, sel1_n: std_logic; begin sel0_n <= not sel0; sel1_n <= not sel1; d0 <= sel0_n and sel1_n ; d1 <= sel0 and sel1_n; d2 <= sel0_n and sel1; d3 <= sel0 and sel1; end behavS; </pre>

Figura 2.78 Multiplexor y decodificador. Descripción mediante sentencias de asignación de señal simples.

Elaboración teniendo en cuenta una familia de FPGA (“Post-Fitting”⁴⁴). Quartus también muestra, además de la elaboración RTL, un esquema del circuito después de ser sintetizado para una familia de FPGA. Además del esquema de circuito incluye detalles de las celdas lógicas (“logic cells”) que se utilizar para implementar las distintas partes del circuito. En la Figura 2.79 se muestran las elaboraciones de un multiplexor.

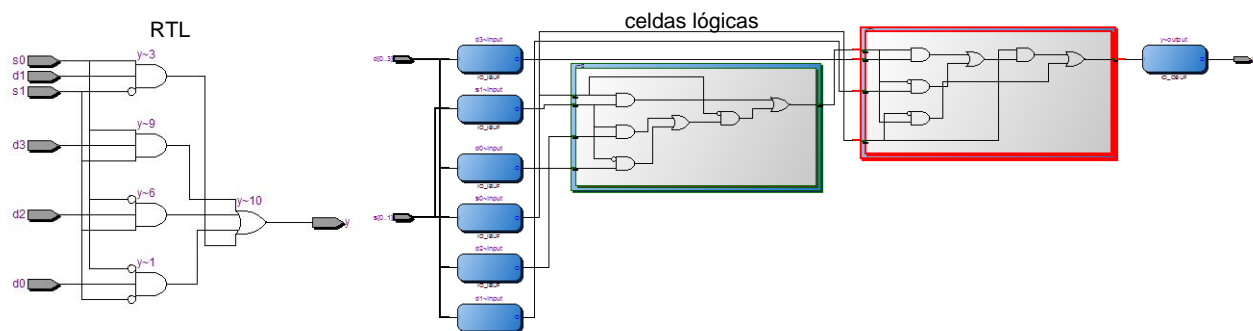


Figura 2.79 Elaboraciones RTL y de celdas lógicas (síntesis) de un multiplexor.

En la Figura 2.80 se muestran las elaboraciones de un decodificador.

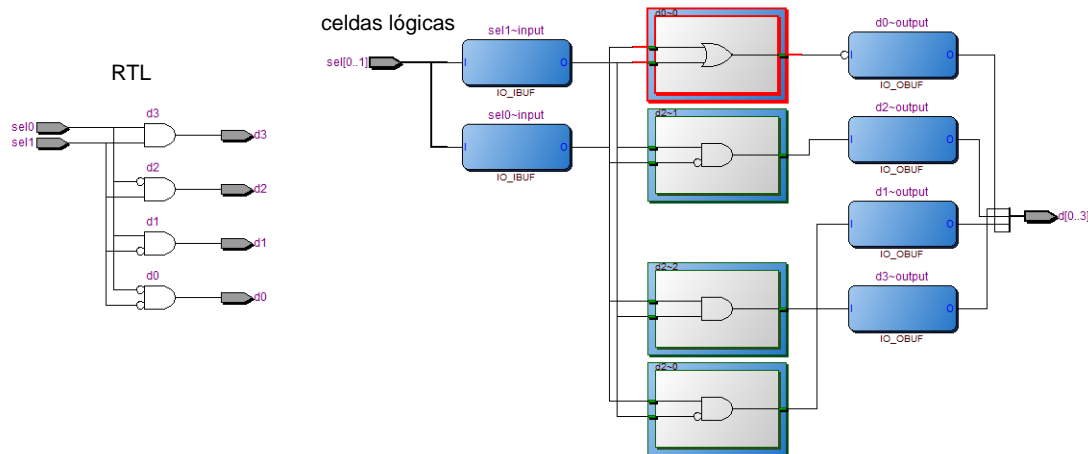


Figura 2.80 Elaboraciones RTL y de celdas lógicas (síntesis) de un decodificador.

Podemos obtener especificaciones más compactas del multiplexor si utilizamos conversión de tipos. En concreto convertimos un objeto de tipo `std_logic_vector` en un objeto de tipo `integer` para valores positivos (sin signo) utilizando la función `to_integer` (Figura 2.81).

```

Multiplexor
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mux4 is
port (d: in std_logic_vector (3 downto 0);
      s: in std_logic_vector (1 downto 0);
      y: out std_logic);
end mux4;

architecture behav of mux4 is
begin
    y <= d(to_integer(unsigned(s)));
end behav;
  
```

Figura 2.81 Descripción VHDL de un multiplexor utilizando un índice para acceder a un elemento de un bus.

En la Figura 2.80 se muestran las elaboraciones de un multiplexor que puede ser parametrizado (mux4).

44. En este curso no se efectúa síntesis. En consecuencia, el interés está en la elaboración RTL. La elaboración en celdas lógicas de se añade por completitud y debido a que, en ocasiones, en la elaboración RTL Quartus muestra un módulo que es una caja negra.

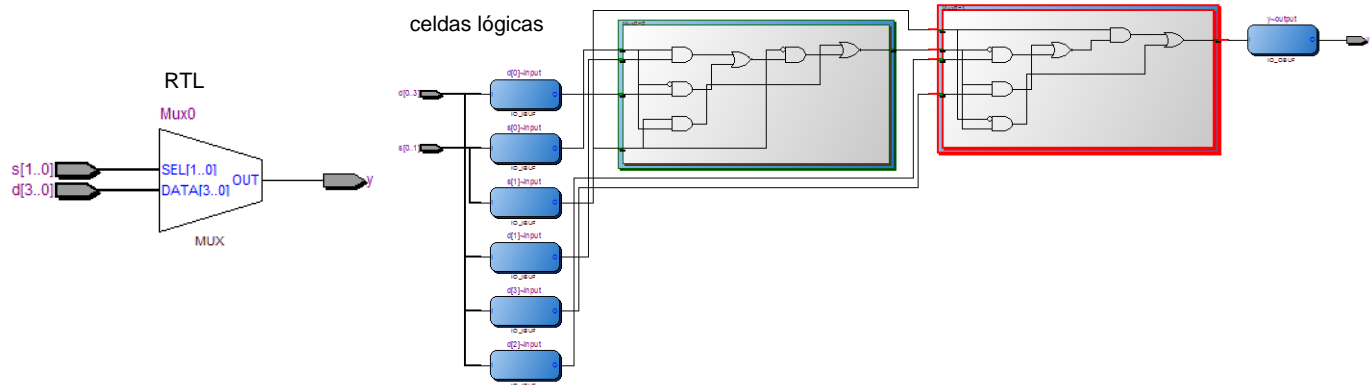


Figura 2.82 Elaboraciones RTL y de celdas lógicas (síntesis) de mux4.

Asignación de una señal de forma condicional

Una sentencia de asignación concurrente de señal asocia una señal a una expresión. El término, asignación de una señal de forma condicional, se utiliza para describir sentencias donde una señal puede tener asociada más de una expresión. Cada una de las expresiones está asociada con una condición concreta.

La sintaxis de una sentencia de asignación de señal condicional es la siguiente.

Sintaxis
Target_signal <= expression when Boolean_expression else
expression when Boolean_expression else
...
expression;

Las condiciones se evalúan en secuencia y la señal recibe el valor de la primera expresión cuya condición lógica (Booleana) se cumple ("true"). Si no se cumple ninguna condición, la señal recibe el valor de la última expresión. Como las condiciones se evalúan en secuencia, si se cumple más de una condición, el valor que se asigna a la señal es el de la expresión asociada a la primera condición que se cumple. Una condición se basa en el estado de otras señales en el circuito.

Observemos que existe sólo un operador de asignación asociado con una sentencia de asignación condicional de señal.

La asignación de señal condicional se reevalúa tan pronto como cualquiera de las señales en las condiciones o expresiones cambia. El constructor "when-else" es útil para expresar funciones lógicas en forma de tabla de verdad.

Mediante operadores de relación se construyen expresiones cuyo resultado es de tipo boolean. El tipo entero y los tipos enumerados de boolean y caracteres están ordenados.

Operadores relacionales. “=” y “/=” entre otros. El primero es el operador igual y el segundo es el operador no igual o distinto. En particular, estos dos operadores se pueden utilizar para cualquier tipo de datos.

En la Figura 2.83 se muestran dos ejemplos de un multiplexor descrito con sentencias de asignación condicional de señales. La diferencia entre las dos descripciones reside en la utilización de señales binarias o de un bus en la selección de la entrada. Así mismo, en el caso de la descripción VHDL de la parte derecha, las señales de entrada son buses y el número de señales que transporta el bus se especifica de forma genérica. En la descripción se utiliza el operador relacional “=” para establecer condiciones. Observemos que en el caso de buses el valor se especifica utilizando dobles comillas como se ha comentado previamente.

Notemos que la especificación VHDL en los dos casos mostrados en la Figura 2.83 son una traducción directa de la tabla de verdad de un multiplexor. En el caso del código mostrado a la izquierda de la Figura 2.83 la condición “sel1=1 and sel0=1” no se explicita, ya que si no se cumple ninguna condición se selecciona la última expresión.

Señales de 1 bit	Buses
<pre> entity mux is port (d0, d1, d2, d3: in std_logic; sel1, sel0 : in std_logic; Y: out std_logic); end mux; architecture behC of mux is begin Y <= d0 when sel1='0' and sel0='0' else d1 when sel1='0' and sel0='1' else d2 when sel1='1' and sel0='0' else d3; end behC; </pre>	<pre> entity muxN is generic (n: positive := 4); port (d0, d1, d2, d3: in std_logic_vector (n-1 downto 0); SEL: in std_logic_vector (1 downto 0); Y: out std_logic_vector (n-1 downto 0)); end muxN; architecture behCG of muxN is begin Y <= d0 when SEL = "00" else d1 when SEL = "01" else d2 when SEL = "10" else d3; end behCG; </pre>

Figura 2.83 Multiplexor. Descripción mediante asignación de señal de forma condicional.

En la Figura 2.84 se muestra la elaboración RTL de mux.

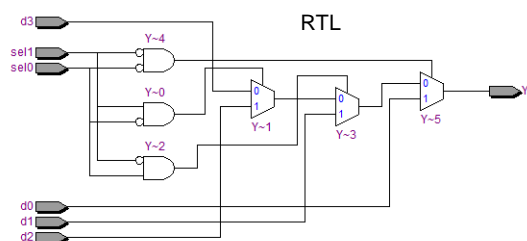


Figura 2.84 Elaboración RTL de mux.

En la Figura 2.85 se muestra la elaboración con celdas lógicas de mux.
celdas lógicas

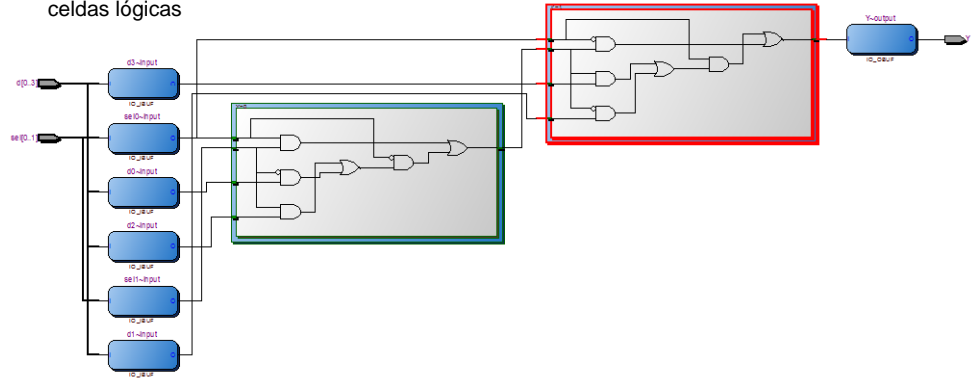


Figura 2.85 Elaboración con celdas lógicas (síntesis) de mux.

En la Figura 2.86 se muestra la elaboración RTL de muxN.

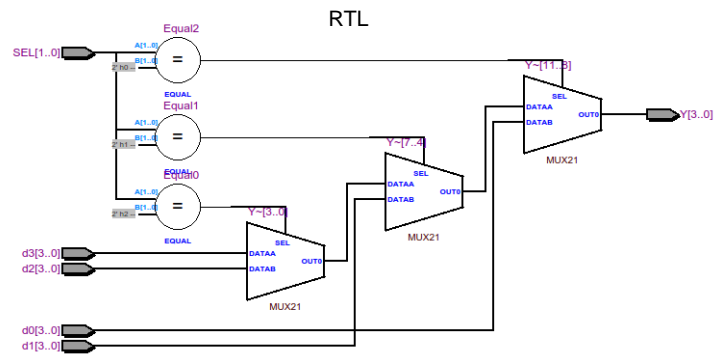


Figura 2.86 Elaboración RTL de muxN.

En la Figura 2.87 se muestra la elaboración con celdas lógicas de muxN.

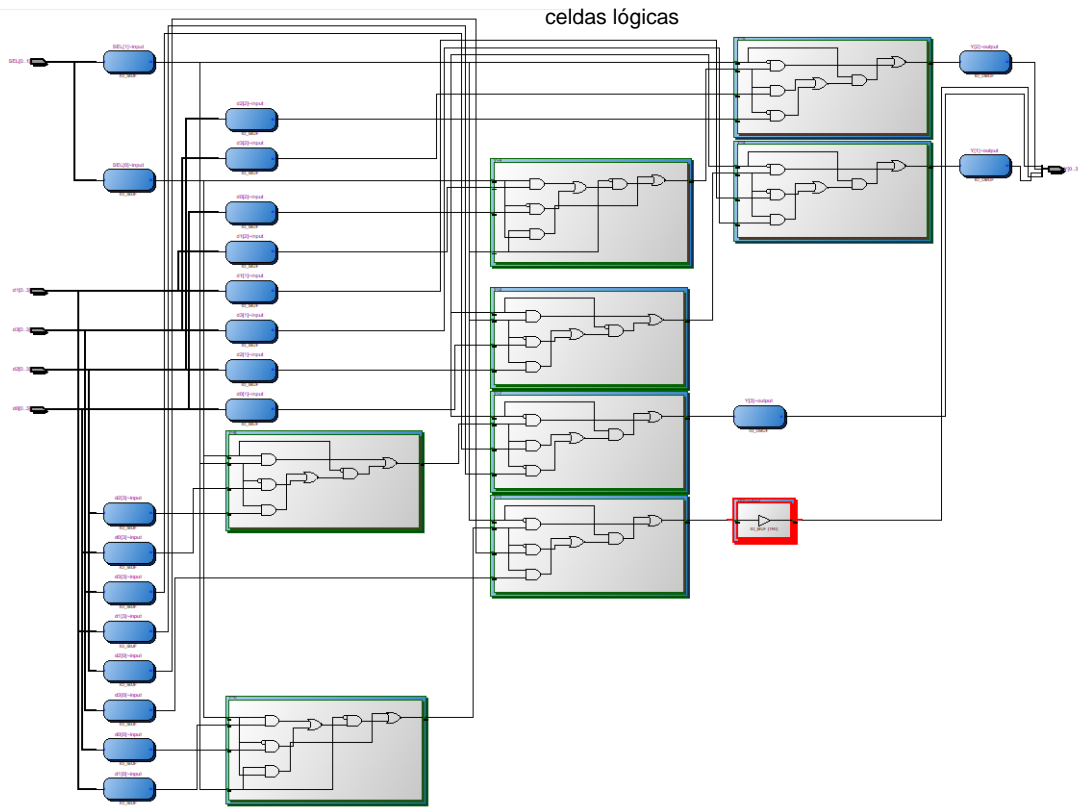


Figura 2.87 Elaboración con celdas lógicas (síntesis) de muxN.

En la parte izquierda de la Figura 2.88 se muestra la especificación VHDL de un decodificador de 3 entradas para 8 salidas. Notemos que igual que en el caso del multiplexor la especificación es una traducción directa de la tabla de verdad. En este caso, para tener en cuenta valores de las señales distintos de 1 y 0, se han especificado todas las condiciones de la tabla de verdad. La última expresión "xxxxxxx" será seleccionada cuando el valor de la señal de selección no es uno de los valores especificados previamente.

En la parte derecha de la Figura 2.88 se muestran la especificación VHDL de una puerta de tres estados. El número de señales del bus de entrada se especifica de forma genérica. Por ello, para especificar alta impedancia en el caso de que la señal permiso ("pe") tome el valor cero se utiliza "others".

Decodificador

Puerta de tres estados

Sumador de 4 bits

```

library IEEE;
use IEEE.std_logic_1164.all;

entity decodificadorC is
port (Sel : in std_logic_vector (2 downto 0);
      D: out std_logic_vector (7 downto 0) );
end decodificadorC;

architecture behC of decodificadorC is
begin
    D <= "00000001" when Sel="000" else
        "00000010" when Sel="001" else
        "00000100" when Sel="010" else
        "00001000" when Sel="011" else
        "00010000" when Sel="100" else
        "00100000" when Sel="101" else
        "01000000" when Sel="110" else
        "10000000" when Sel="111" else
        "xxxxxxx";
end behC;

```

```

library IEEE;
use IEEE.std_logic_1164.all;

entity tresestadosW is
generic (n : positive := 4);
port (pe : in std_logic;
      en: in std_logic_vector (n-1 downto 0);
      sal: out std_logic_vector (n-1 downto 0) );
end tresestadosW;

architecture behavT of tresestadosW is
begin
    sal <= en when (pe = '1') else (others => 'z');
end behavT;

```

Figura 2.88 Decodificador. Descripción mediante asignación de señal de forma condicional.

En la Figura 2.89 se muestra la elaboración RTL del decodificadorC.

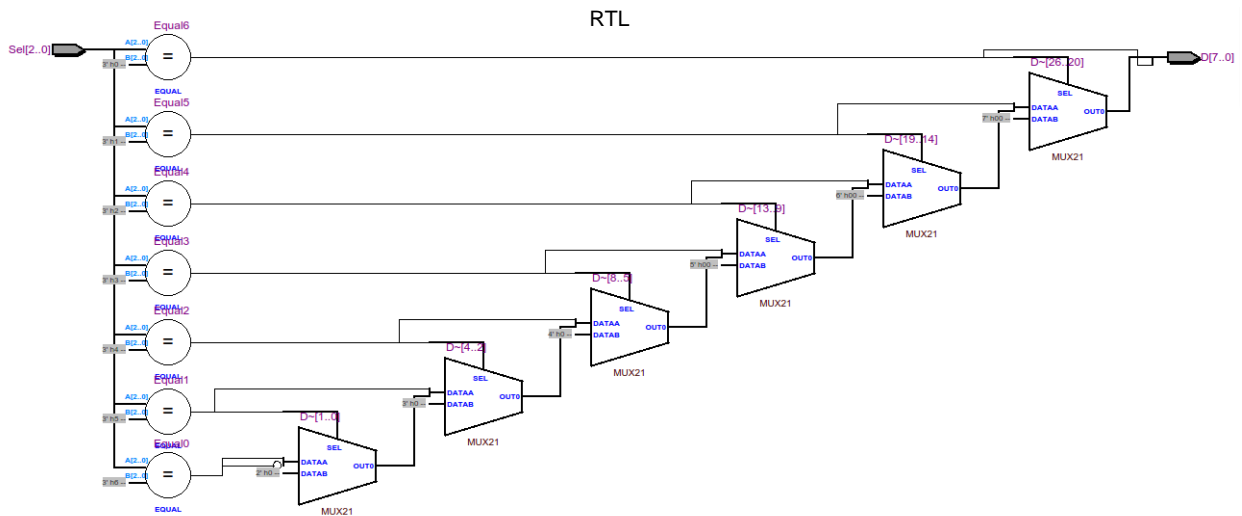


Figura 2.89 Elaboración RTL del decodificadorC.

En la Figura 2.90 se muestra la elaboración con celdas lógicas del decodificadorC.

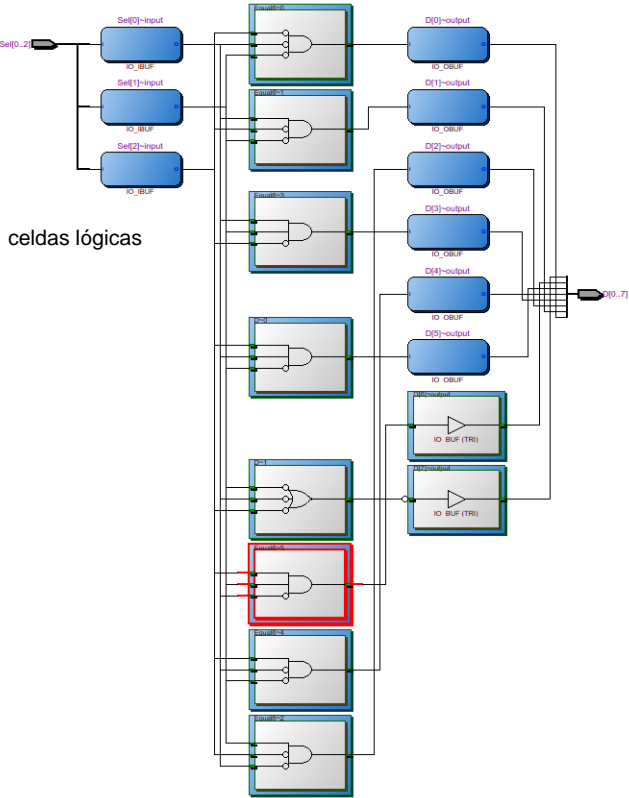


Figura 2.90 Elaboración con celdas lógicas (síntesis) del decodificadorC.

En la Figura 2.91 se muestra la elaboración RTL y con celdas lógicas de tresestadosW.

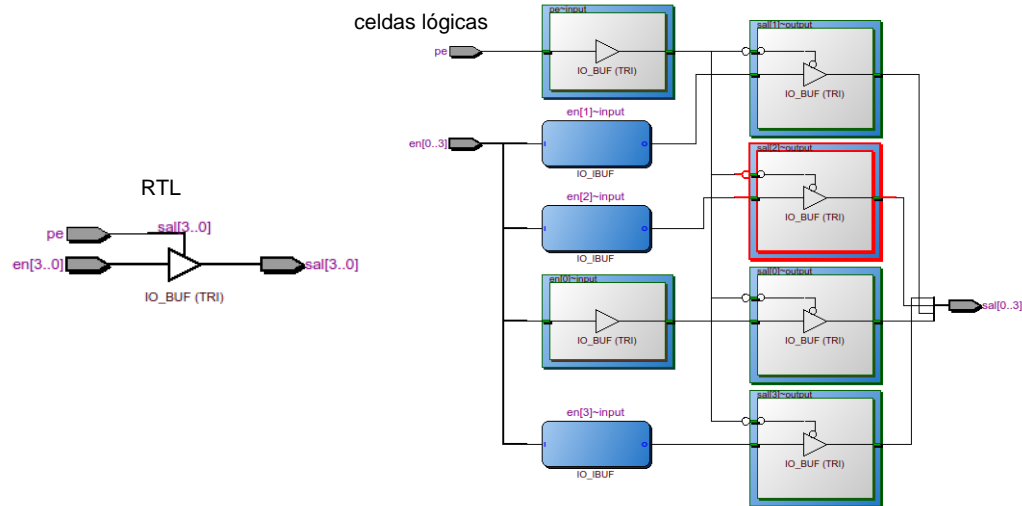


Figura 2.91 Elaboraciones RTL y con celdas lógicas de tresestadosW.

Selección en la asignación de una señal

Una asignación de señal con selección es similar a una selección condicional de la señal. Estas sentencias pueden tener sólo una señal a la que se asigna un valor y únicamente una expresión determina cuál de las elecciones es la asignada. La sintaxis es la siguiente.

Sintaxis
<pre>[label:] with choice_expression select target_name <= expression when choices, ... expression when choices;</pre>

La sentencia de asignación de forma selectiva se evalúa cada vez que existe un cambio en “choice_expression”.

La señal recibirá el valor de la expresión cuya elección (“choices”) está incluida en los valores de la expresión de elección (“choice_expression”), los cuales son constantes. La expresión seleccionada es la primera con una coincidencia en la elección. La elección puede ser una expresión fija (ej. 3) o una expresión que identifica un rango (ej. 3 to 12). Para una elección se siguen las siguientes reglas:

- Dos posibles elecciones tienen que tener una intersección nula.
- Todos los posibles valores de la expresión de elección deben estar presentes en el conjunto de elecciones, a menos que la palabra clave “others” esté presente como elección.

Las elecciones pueden expresar un único valor, un rango o elecciones combinadas. Seguidamente se muestra un ejemplo. En la segunda elección se incluyen varios posibles valores. La elección “others” debe ser la última de las elecciones.

Ejemplo

```
target <= value1 when "000",
        value2 when "001" | "011" | "101",
        value3 when others;
```

En el ejemplo anterior la barra vertical “|” se utiliza como un carácter de selección en la sección de “choices” de una sentencia de selección.

En la Figura 2.92 se muestran dos ejemplos de un multiplexor descrito con sentencias de asignación condicional de señal. La diferencia entre las dos descripciones reside en la utilización de un bus o de valores codificados (enteros) en la selección de la entradas. Así mismo, en el caso de la descripción VHDL de la parte derecha, las señales de entrada son buses y el número de señales que transporta el bus y el rango de la codificación se especifican de forma genérica. Notemos que la especificación VHDL es una traducción directa de la tabla de verdad.

Selección mediante vector de bits	Buses. Selección mediante números codificados
<pre>library IEEE; use IEEE.std_logic_1164.all; entity muxW is port (d0, d1 : in std_logic; d2, d3 : in std_logic; s : in std_logic_vector (1 downto 0); y: out std_logic); end muxW; architecture behavS of muxW is begin with s select y <= d0 when "00", d1 when "01", d2 when "10", d3 when "11"; end behavS;</pre>	<pre>library IEEE; use IEEE.std_logic_1164.all; entity muxNW is generic (n: positive := 4; m: positive := 4); port (d0, d1: in std_logic_vector (m-1 downto 0); d2, d3: in std_logic_vector (m-1 downto 0); s : in integer range 0 to n-1; y: out std_logic_vector (m-1 downto 0)); end muxNW; architecture behavSG of muxNW is begin with s select y <= d0 when 0, d1 when 1, d2 when 2, d3 when 3; end behavSG;</pre>

Figura 2.92 Multiplexor. Descripción mediante selección en la asignación de una señal

En la Figura 2.93 se muestra la elaboración RTL y con celdas lógicas de muxW.

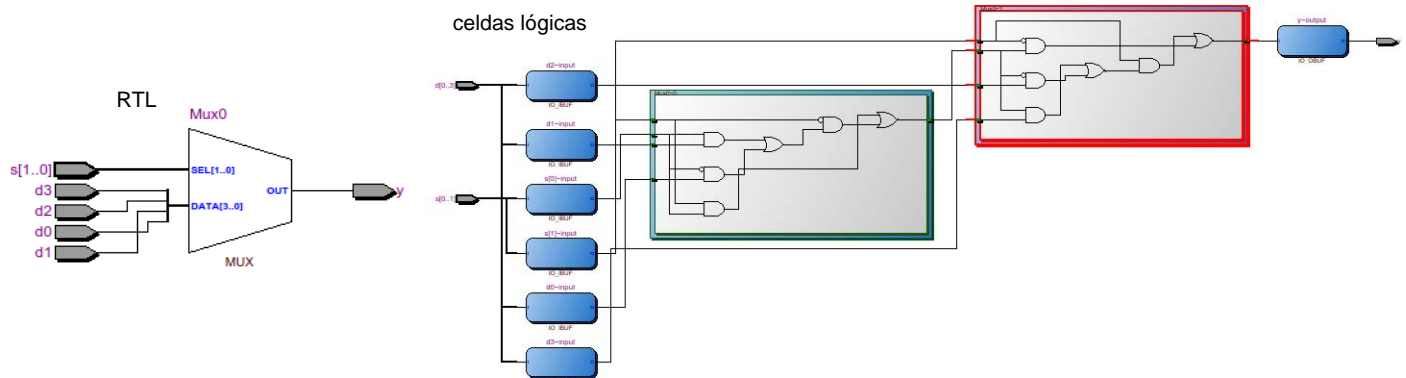


Figura 2.93 Elaboraciones RTL y con celdas lógicas de muxW.

En la Figura 2.94 se muestra la elaboración RTL muxNW.

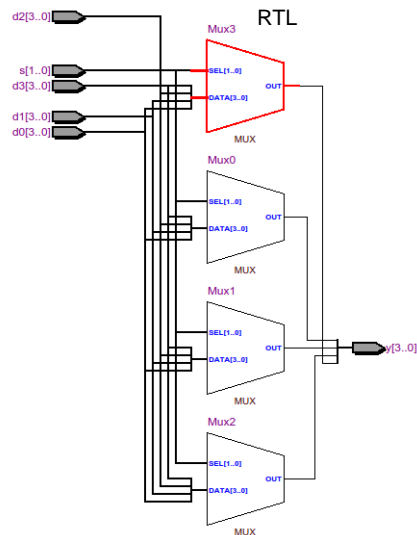


Figura 2.94 Elaboración RTL de muxNW.

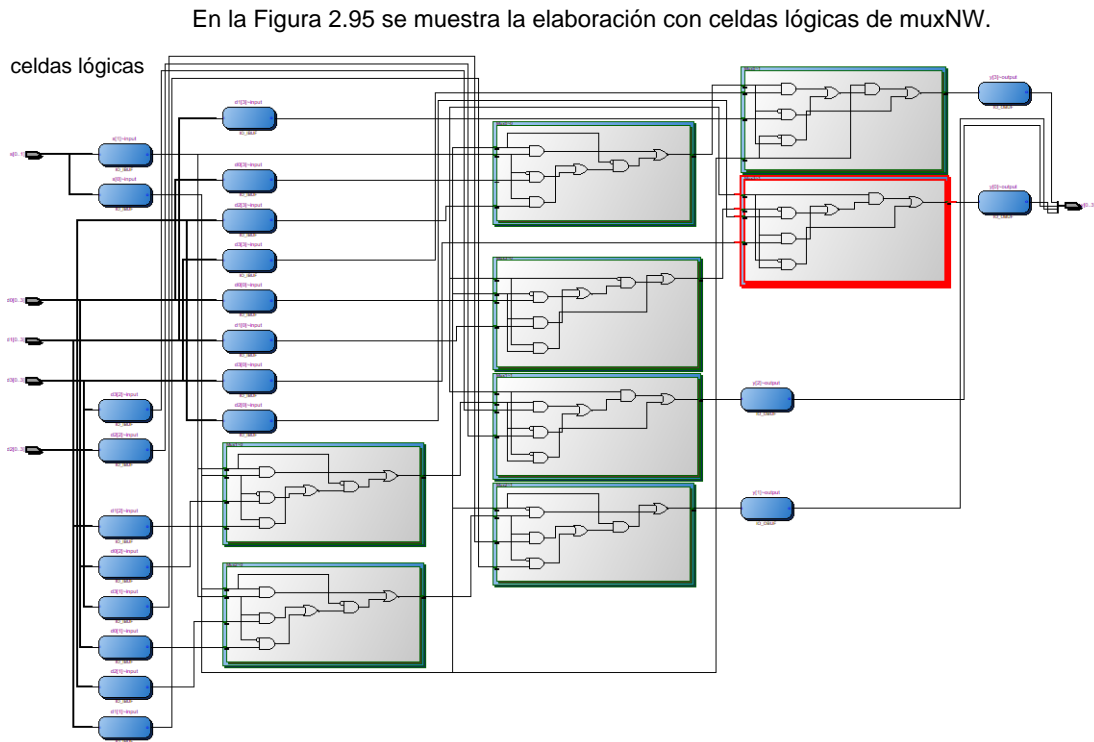


Figura 2.95 Elaboración con celdas lógicas de muxNW.

En la parte izquierda de la Figura 2.96 se muestra la especificación de un decodificador de 3 entradas para 8 salidas. En esta especificación se tiene en cuenta el caso de que alguna de las señales del bus de selección no tome el valor 1 o 0.

En la parte derecha de la Figura 2.96 se muestran la especificación VHDL de una puerta de tres estados. El número de señales del bus de entrada se especifica de forma genérica. Por ello, para especificar alta impedancia, en el caso de que la señal permiso ("pe") tome el valor cero, se utiliza "others".

Bus de salida	Puerta de tres estados
<pre>library IEEE; use IEEE.std_logic_1164.all; entity decodificadorS is port (Sel : in std_logic_vector (2 downto 0); D: out std_logic_vector (7 downto 0)); end decodificadorS; architecture behS of decodificadorS is begin with Sel select D <= "00000001" when "000", "00000010" when "001", "00000100" when "010", "00001000" when "011", "00010000" when "100", "00100000" when "101", "01000000" when "110", "10000000" when "111" ; "00000000" when others ; end behS;</pre>	<pre>library IEEE; use IEEE.std_logic_1164.all; entity tresestadosS is generic (n : positive := 4); port (pe : in std_logic; en: in std_logic_vector (n-1 downto 0); sal: out std_logic_vector (n-1 downto 0)); end tresestadosS; architecture behavT of tresestadosS is begin with pe select sal <= en when '1', (others => 'z') when '0'; end behavT;</pre>

Figura 2.96 *Decodificador. Descripción mediante selección en la asignación de una señal.*

En la Figura 2.97 se muestra la elaboración RTL y con celdas lógicas de decodificadores.

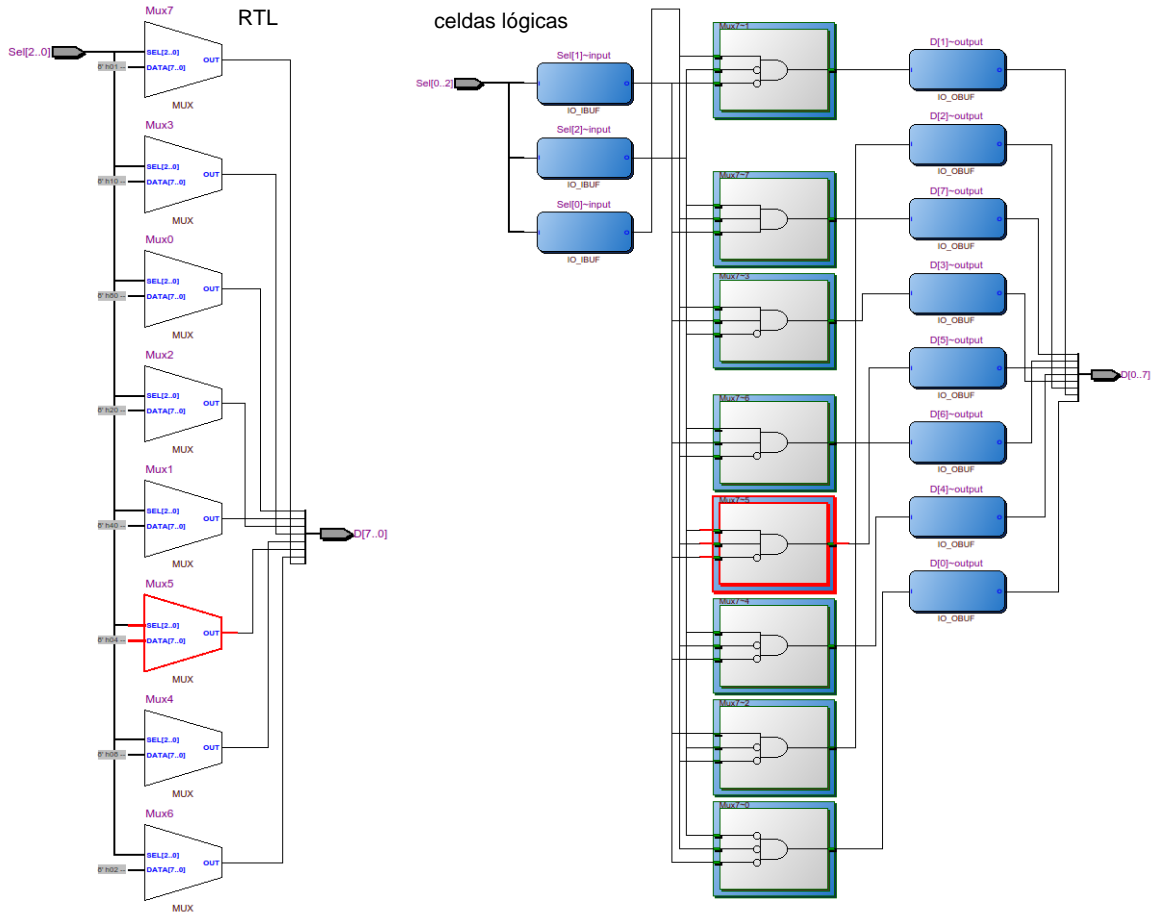


Figura 2.97 Elaboraciones RTL y con celdas lógicas de decodificadores.

En la Figura 2.98 se muestra la elaboración RTL y con celdas lógicas de tresestados.

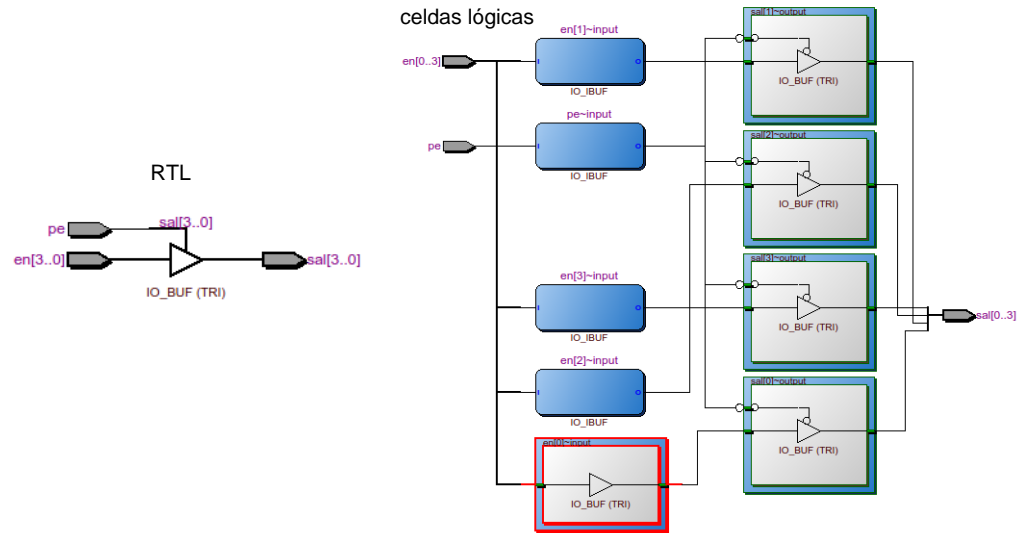


Figura 2.98 Elaboraciones RTL y con celdas lógicas de tresestados.